Titre: Title:	Verification approach for compositional hierarchical time Petri-nets
Auteur: Author:	Lan Chen
Date:	2005
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Chen, L. (2005). Verification approach for compositional hierarchical time Petri-nets [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7635/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/7635/
Directeurs de recherche: Advisors:	
Programme: Program:	Non spécifié

UNIVERSITÉ DE MONTRÉAL

VERIFICATION APPROACH FOR COMPOSITIONAL HIERARCHICAL TIME PETRI-NETS

LAN CHEN
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2005

Your file Votre référence
ISBN: 978-0-494-16805-9
Our file Notre référence
ISBN: 978-0-494-16805-9

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

VERIFICATION APPROACH FOR COMPOSITIONAL HIERARCHICAL TIME PETRI-NETS

présenté par : CHEN Lan

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. QUINTERO Alejandro, Doct., président

Mme BOUCHENEB Hanifa, Doctorat, membre et directrice de recherche

Mme NICOLESCU Gabriela, Doct., membre

ACKNOLEDGEMENTS

I wish to express my deepest appreciation and gratitude to my supervisor, professor BOUCHENEB Hanifa, for her guidance, encouragement, inspiration, valuable interaction and critical suggestions that greatly helped in improving the quality of this work during my years of graduate study. I thank her for her immense patience in reading, correcting this work with her thoughtful insights.

I also appreciate Mr. Rachid Hadjidj, a Ph.d candidate at the Vérification des systèmes temps reel Research Lab, for all his help of providing original version of the useful software which help to promote my research moving advance when it is in a puzzledom. The Graphical tool I select to modify and program for my research finally in this project is based on Rachid's previous work.

Special thanks to my dear family member, my parents, ma brothers and my husband, for their great patience, encouragement, love and understanding all through my life, and to them, I dedicate this thesis.

Lastly, I dedicate this thesis to my loveliest one year old son – little Hubert.

ABSTRACT

Although Petri nets (PNs) have for many years been applied as a formal model in which one could describe and analyze systems, it can be argued that they are not widely used in industrial-sized systems. The main reason for this can be attributed to a lack of compositionality which means that PNs were unable to deal with large complex or even midsized systems. A useful idea for this problem is to handle the complexity of systems by composing them out of smaller components. In the past decade, Many attempts to bring compositionality and modularity into Petri nets ([BV95] [SM83] [RS83] [RA96] [ES96] [JA98]) did result in a substantial progress in system compositional modeling. It is crucial for practical application of PNs systems.

There are three primary objectives in this thesis:

- Give a precise notion of composition and investigate, review existing approaches in compositionality area of Petri nets, offer an up-to-date detailed survey of these approaches, classify explicitly composition mechanisms. The reason for this aim is because most of relevant works about composition just focus emphasis on techniques, theoretical points of view, advantages or capacities, practical examples, There are little and sparse attentions of an overview of available composition mechanisms for large Petri nets models.

- Take advantage of recent developments and try to find an available approach among these various composition techniques for Petri nets with time extended

systems. As a result, an effective way will be presented to construct time Petri nets. It provides necessary concepts of compositionality, which are not only necessary for analysis but are obvious requirements for the construction of many sizeable models.

- Implement this compositional structure construction by textual or graphical editor in some software. Petri nets are a widely used modeling technique and a number of Petri net editor tools exist to design Petri nets. However, if no any current tool that ca provide full support for Petri nets or satisfy our all requirements (time constraints definition, composition and decomposition structure legibility; properties equivalence after composing; enumerative method for Time Petri nets), improved and programmed an appropriable software need to be done in this project.

RÉSUMÉ CONDENSÉ EN FRANÇAIS

INTRODUCTION

Bien que les Réseaux de Petri aient été appliqués pendant des années comme un modèle formel pour l'analyse des systèmes, ils n'ont pas été largement utilisés pour la description et l'analyse des systèmes industriels. La raison principale de cette situation est de au manque de compositionnalité signifiante que les réseaux de Petri ont été incapables d'appréhender les systèmes larges, complexes et même de taille moyenne. Ce mémoire a trois objectifs majeurs:

1. Définir précisément la notion de composition des réseaux de Petri, en discuter des approches existantes, présenter l'état de l'art de la littérature sur ces approches, classer les mécanismes de composition.

2. S'inspirer des récents développements et trouver une approche baseé sur les réseaux de Petri temporels. Subséquemment, une méthode effective sera présentée pour la construction de réseaux de Petri.

3. Implémenter cette structure de composition sous forme d'un logiciel de construction munie d'un éditeur textuel ou graphique.

SURVOL ET MOTIVATION

Les réseaux de Petri sont bien indiqués pour la modélisation de systèmes de par leur nature graphique et par la décomposition des états et des événements en transitions et en places. Ces propriétés justifient l'excellence des réseaux de Petri en tant qu'outil pour la validation de modèles.

Cependant, ils sont insuffisants pour la modélisation de systèmes temporels (TPNs) qui ont des architectures complexes et à plusieurs couches. La raison principale est que les TPNs modélisent toujours les systèmes en un réseau plat, et ne possèdent pas de mécanismes permettant la compositionalité et la modularité. Basé sur les études des approches de composition, un modèle étant composé de sous modèles peut être développé. La modélisation hiérarchique peut aider à diminuer le temps alloué à la phase de conception et à la réutilisabilité des composants.

RECHERCHE, OBJECTIFS ET PROCESSUS

Le projet se concentre, d'abord, sur une revue bibliographique des méthodes de composition, de la spécification et du design des systèmes temporels complexes. Celle-ci s'effectue en deux étapes:

- *Faire une revue* et *discuter* des liens existants dans la littérature sur la compositionalité. Décrire les différentes approches, leurs motivations et discuter des avantages et des désavantages de ces travaux. Ensuite, une méthode appropriée sur les réseaux de Petri temporels supportant la composition doit être trouvée.

- *Discuter* le rôle de la théorie Hiérarchique/Abstraite sur les réseaux de Petri temporels et faire un survol des concepts pertinents de la théorie de la composition.

Un autre but de ce projet est de trouver ou de développer un outil graphique adapté ayant comme fonctionnalité l'édition d'intervalles de temps et la vérification du modèle en temps réel. Cet outil doit permettre la représentation de la composition hiérarchique dans les réseaux de Petri temporels avec la possibilité de concevoir des réseaux descendants (top-bottom) et ascendants (bottom-up) et de naviguer à travers un réseau de Petri hiérarchique. Les étapes de ce travail sont:

- *Concevoir et programmer* un outil d'édition de réseaux de Petri qui comblera le manque actuel de la composition dans les réseaux de Petri temporels hiérarchiques et des limitations en termes d'analyses des éditeurs actuels.

- *Implémenter* des réseaux de Petri temporels à l'aide de l'interface usager graphique. Cette implémentation permettra la visualisation des relations entre les composants constituant les réseaux de Petri.

- *Calculer* les classes d'États (SC) et *générer* le Graphe des classes d'États (SCG) ([BD91] [BH04] [BH05]) en utilisant les outils sélectionnés sur les réseaux de Petri ou des outils programmés.

- *Exécuter* des exemples de réseaux de Petri (*Problème des philosophes* et *Systèmes de Manufactures flexibles*) en utilisant la théorie de la composition hiérarchique et les éditer sur l'outil de conception, sélectionné ou développé, de réseaux de Petri.

SYNOPSIS

Le plan de ce mémoire est le suivant: Le **chapitre 2** définit les concepts utilisés, la validation du modèle, notamment les techniques permettant d'analyser les réseaux de Petri temporels. Le **chapitre 3** décrit la technique de composition hiérarchique comme méthode pour la résolution de la conception de systèmes complexes telle que mentionnée dans la revue de littérature. Dans le **chapitre 4,** les codes de l'implémentation de l'interface usager graphique et de la composition des réseaux de Petri temporels sont présentés. Le **chapitre 5** traite des études d'un cas industriel complexe de systèmes de manufactures flexibles (FMS). Le graphe des classes d'états est généré pour chaque cas et les résultats seront analysés. Le **chapitre 6** conclut ce mémoire en soulignant les contributions, les limitations et les suggestions de travaux futurs.

RÉSEAUX DE PETRI TEMPORELS - CONNAISSANCES PRÉLIMINAIRES

Les réseaux temporels étendent les réseaux de Petri en associant des spécifications temporelles. Il y a trois extensions principales pour les réseaux de Petri. Dans ce projet, nous introduisons uniquement les réseaux de Petri temporels de Merlin en détail (voir figure 1).

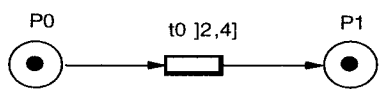


Figure 1: Réseaux de Petri temporels de Merlin

Définition des réseaux de Petri temporels de Merlin (TPNs): Comme le montre l'exemple à la figure 1, un TPN est formellement définie comme un 6-tuple

TPN $= < P, T, I, O, M_0, SI >$ où:

- (P, T, I, O, M_0) est un réseau de Petri classique. (Où les transitions T ont un temps de délai minimum et maximum de tir)
- SI est une fonction. $SI : T \rightarrow Q^+ \times (Q^+ \cup \infty)$, où Q est l'ensemble de nombres rationnels

Pour analyser les TPNs, il y a plusieurs approches telles que: *L'analyse Invariante, L'analyse de Convergence (Reachability Analysis), L'énumération d'états (State Enumeration), L'évaluation de Performance (Performance Evaluation), La simulation de Jeux de Jetons (Token Game Simulation).* Dans ce projet, l'emphase sera portée sur l'énumération d'états. La méthode du Graphe de Classes d'États **(SCG)** pour analyser les TPNs est une instance de l'énumération d'états. Un graphe de Classes d'États (SCG) est un graphe de toutes les classes d'états. Par exemple, considérons le modèle de TPNs illustré à la Figure 2, nous obtenons les classes d'états et le graphe des classes d'états ci-dessous (Voir Figure 3 et Figure 4).

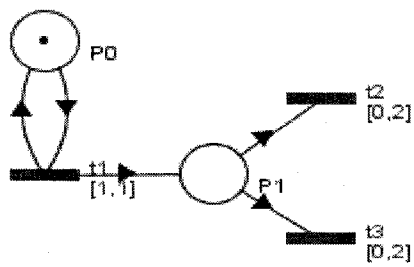


Figure 2: Modèle d'un simple réseau de Petri temporel

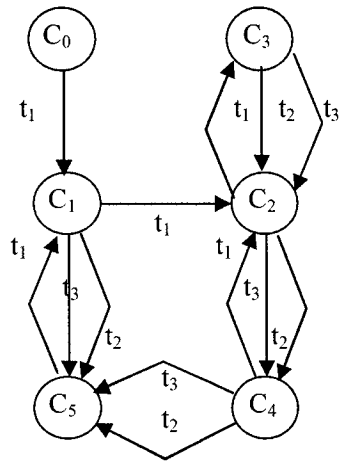


Figure 3: Graphe de classes d'états du TPNs de la Figure 2

(Initial class)C_0 $M_0=p_0$; $D_0=\{1\leq\theta(t_1)\leq1\}$	C_1 $M_1=p_0,p_1$; $D_1=\{1\leq\theta(t_1)\leq1,$ $0\leq\theta(t_2)\leq2,$ $0\leq\theta(t_3)\leq2\}$	C_2 $M_2=p_0,p_1(2)$; $D_2=\{1\leq\theta(t_1)\leq1,$ $0\leq\theta(t_2^0)\leq1,$ $0\leq\theta(t_2^1)\leq2,$ $0\leq\theta(t_3^0)\leq1,$ $0\leq\theta(t_3^1)\leq2\}$
C_3 $M_3=p_0, p_1(3)$; $D_3=\{1\leq\theta(t_1)\leq1,$ $0\leq\theta(t_2^0)\leq0,$ $0\leq\theta(t_2^1)\leq1,$ $0\leq\theta(t_2^2)\leq2,$ $0\leq\theta(t_3^0)\leq0,$ $0\leq\theta(t_3^1)\leq1,$ $0\leq\theta(t_3^2)\leq2\}$	C_4 $M_4=p_0, p_1$; $D_4=\{1\leq\theta(t_1)\leq1,$ $0\leq\theta(t_2)\leq2,$ $0\leq\theta(t_3)\leq2,$ $\theta(t_2)-\theta(t_1)\leq1,$ $\theta(t_3)-\theta(t_1)\leq1\}$	C_5 $M_5=p_0$; $D_5=\{1\leq\theta(t_1)\leq1\}$

Figure 4: Table de Graphe de classes d'états du TPNs de la Figure 3

MODÈLE LOGIQUE TEMPOREL DE VÉRIFICATION

Le modèle logique temporel de vérification est une technique automatique et formelle pour prouver la validité du fonctionnement d'un TPNs. Nous formalisons seulement les langages de spécification Logique arborée de computation (*Computation Tree logic* CTL) et *Logique temporelle linéaire* (LTL).

CLASSIFICATION DE LA COMPOSITION ET COMPARAISON

De nombreuses approches et mécanismes de structure permettant la compositionalité des modèles existent. Il est nécessaire d'analyser la construction de n'importe qu modèle d'une certaine taille. Avant de discuter des applications pratiques sélectionnées dans ce projet, c'est-à-dire la Compositionnalité des composants, nous devons d'abord formellement décrire les concepts de *Connecteur, Composant (Subnet, Supernet), Fusion, Enveloppement (Folding), Abstrait/ Atomique, Hiérarchie/Raffinement, Algèbre, Orientation Objet.*

Ce mémoire propose une classification des mécanismes des réseaux de Petri et discute chacun d'eux. Ces mécanismes sont constitués de fusion de noeuds (place ou transition), de partage de nœuds (place ou transition), des réseaux de Petri inspirés par l'orienté objet, l'abstraction/raffinement, l'algèbre, le flot de travail, la théorie de la trace et d'autres. Plusieurs articles de revue représentatifs sont utilisés pour souligner

l'application des mécanismes présentés aux champs spécifiques, à savoir la modélisation des systèmes, et le génie logiciel dans les systèmes normaux ou temporels, où une forme de modélisation de mécanismes joue un rôle majeur. Il y a plusieurs approches pour structurer les réseaux de Petri compositionnels selon la revue de littérature. Ces mécanismes sont classés ci-dessous (voir Figure 5)

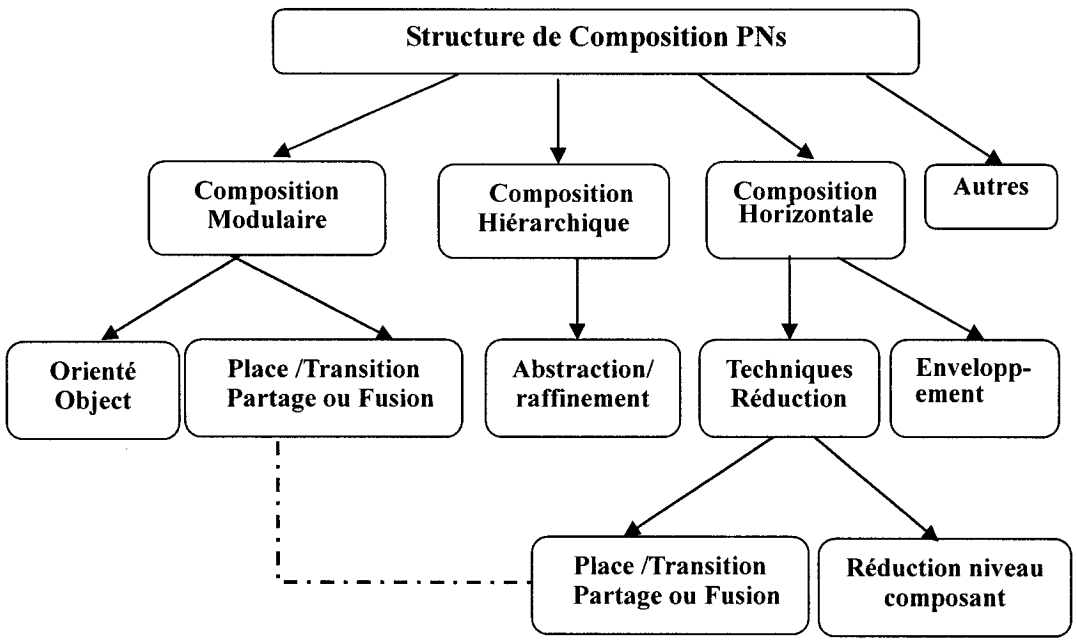


Figure 5: Classification de structure de réseaux de Petri compositionnels

RÉSEAUX DE PETRI TEMPORELS HIÉRARCHIQUES

Dans ce projet, un mécanisme de composition hiérarchique est sélectionnée pour que le modèle soit construit de façon structurée et composé d'unités plus simples facilement compréhensibles par le concepteur à chaque niveau de description (Figure 6). Les concepts hiérarchiques supportent deux mécanismes importants qui permettent

le partage et la reutilisabilité des composants. Ces deux mécanismes sont:

Abstraction/Raffinement (Descendant): Les entités (Place, Transitions) devraient représentées l'information abstraite à un haut niveau de structure de décomposition. En utilisant un concept abstrait, nous pouvons focaliser sur les aspects essentiels et intrinsèques d'une entité et ignorer ses propriétés accidentelles ; *Atténuation (Ascendant)*: À un niveau inférieur, cette abstraction doit être raffiné.

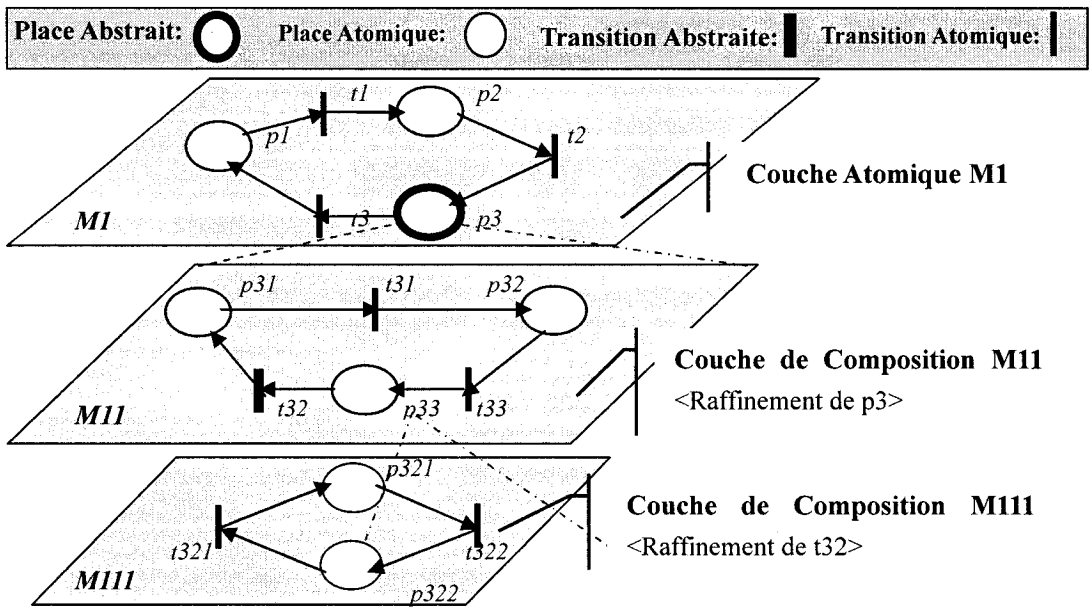


Figure 6: Illustration des Réseaux de Petri hiérarchiques et compositionnels

DÉVELOPPEMENT D'OUTIL ET IMPLÉMENTATION

Nous introduisons l'outil *GraphLab* pour réaliser la construction de structure et la vérification des modèles pour les réseaux de Petri temporels dans ce projet. *GraphLab* est une boîte à outils pour l'analyse des réseaux de Petri (temporels) qui

construit les graphes des classes d'états (modèles abstraits) et qui les exploite pour LTL, CTL. La version originale de l'outil a été développée au laboratoire de recherche de Vérification des systèmes temps réel de École Polytechnique de Montréal. Ce projet a modifié et a ajouté de nouvelles fonctionnalités dans l'outil *GraphLab*. *GraphLab* délivre les fonctions de bases nécessaires au design et à l'édition de réseaux de Petri. Par exemple, la génération d'un graphe de classes d'états illustré (Figure 7) est d'une grande importance pour le projet.

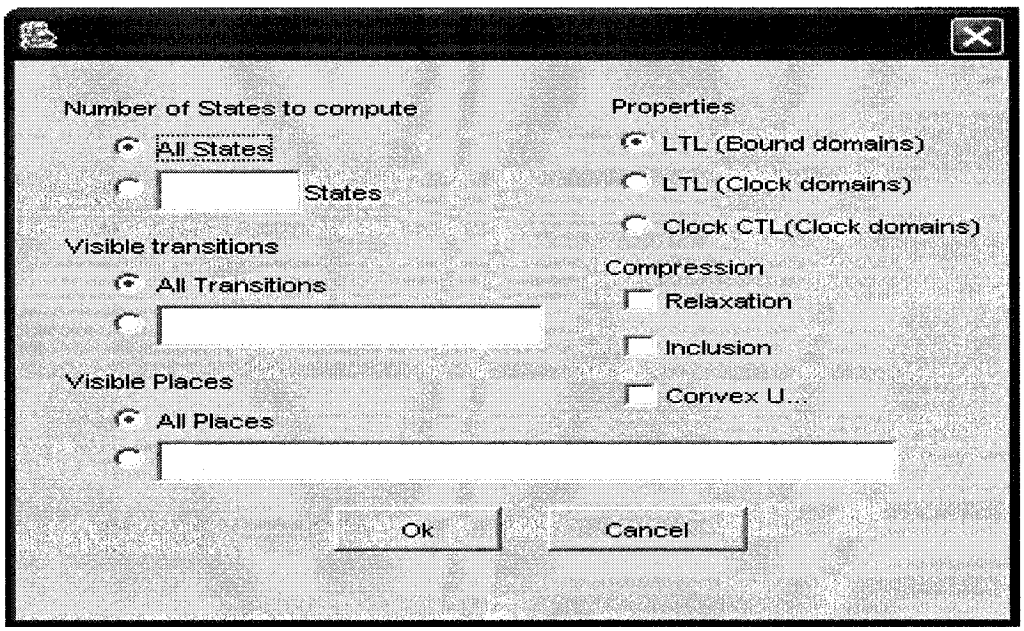


Figure 7: Des propriétées de la Logique Linéaire Temporelle (LTL) pour le graphe de classes d'états

GraphLab a été modifié et augmenté avec de nouvelles fonctions pour supporter la création de réseaux de Petri et la génération de Graphe de Classes d'États. Les modifications de *GrapLab* ont été implémentées dans les langages de programmation JAVA et XML dans l'environnement Jbuilder. Ces modifications sont constituées de:

1. Modifications de l'Interface Graphique Usager et de la composition hiérarchique.

   Telles que: : Édition ou affichage de la structure hiérarchique du *subnet*

ajoutée sur la barre d'outils. : Édition ou affichage de la structure hiérarchique du *supernet* ajoutée sur la barre d'outils.

2.Modification du Design du format de fichier et création de répertoire :

GraphLab utilise un simple format XML pour spécifier les places, les transitions, les arcs, les jetons et les contraintes temporelles. Dans ce projet, l'exemple du problème des Philosophes est utilisé pour montrer la comparaison entre la conception d'un réseau plat et d'un réseau hiérarchique. Lors de l'implémentation de la compositionnalité hiérarchique sur *GraphLab* illustré à la figure 8, nous obtenons la première couche de la structure. Celle-ci contient 5 transitions abstraites.

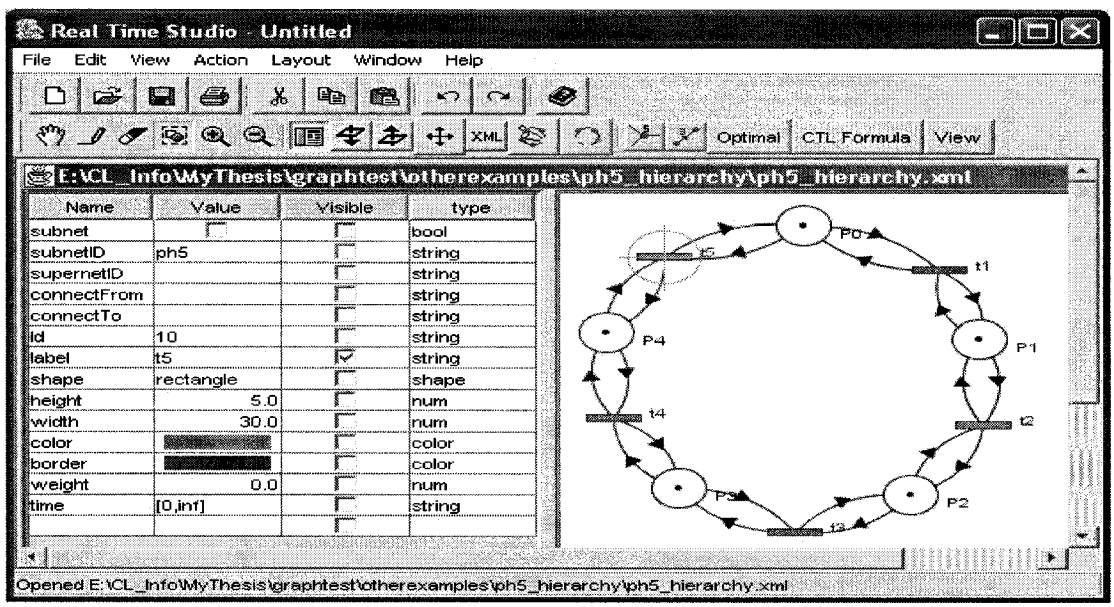


Figure 8: Structure Hiérarchique du problème des Philosophes et Afficheur des attributs

Chacune des 5 transitions abstraites (t1,t2,t3,t4,t5) contient des subnets définies pour les statuts de la pensée abstraite, de la prise des baguettes, de l'alimentation et du dépôt des baguettes des 5 philosophes. L'action de sauvegarde à ce niveau créera 5 répertoires. L'utilisateur peut concevoir le fichier du subnet dans le répertoire pertinent.

Les répertoires enfants possèdent un répertoire de même nom. Après la création de répertoire et la construction de toutes les couches hiérarchiques de composition, l'information du Graphe de Classes sera générée (Figure 9).

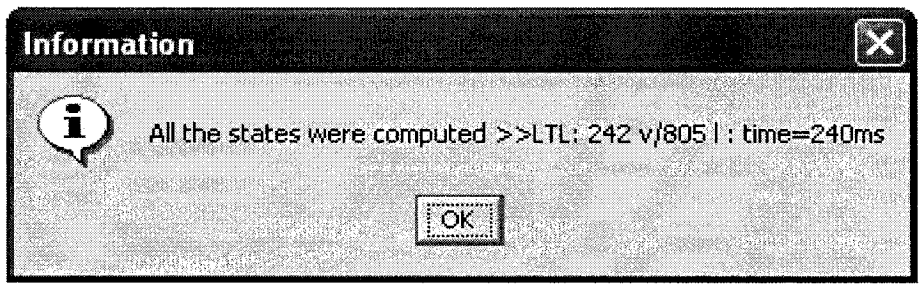


Figure 9: Résultats de la génération du graphe de classes d'états du problème des Philosophes

(242 vertex and 805 liens in state class graph)

L'expérience sur le réseau plat et le design hiérarchique du problème des Philosophes donne le même graphe de classes d'états. Ce résultat illustre les modifications apportées à l'outil *GraphLab* et la faisabilité de l'approche compositionnelle. Enfin, dans ce projet, l'étude d'un exemple de réseaux de Petri hiérarchiques compositionnels dans un système de Manufactures Flexibles (FMS) est utilisé pour investiguer les capacités de l'éditeur Graphique pour l'édition de réseaux de Petri temporels hiérarchiques compositionnels utilisant *GraphLab* selon deux aspects: Donner des descriptions et des résultats expérimentaux. Cela montre comment les sous ensemble (or subnets) d'un système large peut être transformé en utilisant le concept de hiérarchie et l'approche par Graphe de Classes d'États préservant la logique temporelle linéaire.

CONCLUSION ET TRAVAUX FUTURS

Ce travail a présenté les techniques actuelles de composition pour les modèles larges telles que montrée dans la revue de littérature sur les différentes approches compositionnelles.

La synthèse de cette recherche et les contributions apportées sont la démonstration d'une approche utilisant les concepts de hiérarchie et d'abstraction qui sont efficaces pour l'analyse et qui sont des requis évidents pour la construction de modèles temporels étendus –réseaux de Petris temporels. Enfin, un outil logiciel a été développé et amélioré pour la réalisation de cette technique compositionnelle.

Les Limitations sont de l'ordre de la réutilisabilité du design. Si les réseaux de Petri ne peuvent être regroupés dans des parties différentes selon leurs états de tir, et si ces parties (ou subnets) ne peuvent être décomposées de façon descendante, ce large system a une faible efficacité lors de l'utilisation de la composition hiérarchique ascendante.

Finalement, ce mémoire montre que l'analyse incrémentale et l'analyse modulaire sont des éléments importants à investiguer pour des travaux futurs.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

When we create architectural models and verify their properties of complex time-dependent systems, a formal engineering approach that always need to help its modeling, design and analysis. Formal technique - time Petri nets (TPNs) - as a popular modeling tool, provide a rigorous and operational way to describe and analyze system properties [BD91]. In addition to its analytic capability, time Petri nets have been applied to the specification, verification, real-time control and simulation of systems and environments. These strengths make time Petri nets a powerful modeling and verification tool to obtain delays, capacity, reliability measures and deadlock avoidance in complex time-dependent systems.

Like many other formal techniques, the most concern of time Petri nets (TPNs), is the behavior of modeling and analysis, while offering many advantages, however, designing ordinary time Petri nets models suffer from some difficult that limits their usability and application for complex time-dependent systems: Specification time Petri nets in complex systems tend to become too large even for a modest sized model; Lack of mechanisms to structure complex systems integrity; Limited support for graphically editing complex systems via TPNs.

For example, Flexible Manufacturing System (FMS) [WD99] is a typical real-time concurrent system that can be built on top of TPNs to model. Design of the TPNs is the challenge to handle re-constructing, compositionality and refinement techniques which are effective ways let the complex system to form an integrated model and system can be analyzed correctness using Real-Time Logic [AC93].

1.1 Background and Motivation

Petri Nets are well suited for modeling systems with advantages of graphical nature and the decomposition of both states and events into local places and transitions. These properties make Petri nets an excellent tool for the validation of models by non-technical end users. In particular, time Petri nets (TPNs) using timing constraints which are expressed in terms of minimum and maximum amount of time elapsed between the enabling and the execution of each action. This allows a compact representation of the state space and an explicit modeling of a system.

However, despite what we consider to be significant advantages using of TPNs, they are insufficient to cope with the many time-dependent systems that have complex, multilayered architectures. The main reason is ordinary TPNs always model systems in a flat net way, and suffered from a lack of proper support of compositionality and modularity. This results in a failure to deal with reflecting clearly elements that participate in the system and the way they communicate or interact. When reflecting elements that participate in the industrial-sized system communication or interact, specification would often be too heavy to handle efficiently.

In order to manage this complexity and model large complex systems at a relevant level of detail, many attempts of applying different compositionality and modularity into Petri Nets, [AN83] [CO95] [SO91] [JA98] [WD00] did result in a sufficient improvement. Among these approaches includes the various techniques which use a concept of abstraction to allow for structured design and analysis, and fosters composition of model parts such as abstraction based on refining places and transitions [WR90] as well as various object-oriented Petri net approaches (e.g. [ES96] [LA95]).

Based on studies in compositionality, hierarchical composition approach ([VA94] [RA96]) is introduced to construct Petri nets models as a set of components that interact is more appropriate and more important. A model can then be developed as the composition of sub-models that represent components of systems. Hierarchical modeling can help to shorten the time during the design phases. The structure of the resulting model reflects the whole system and these models of components can be developed by different modelers and, in principle, components can be re-used.

1.2 Research Objectives and Processes

This project first concentrates the objective on compositional methods researching for the specification and design to complex time systems, it includes two steps:

- *Review* and *comment* existing literatures that link between compositionality, describing different approaches and motivations and discussing advantages and disadvantages of works studied, and determining what can be learned from each of them. After that, an appropriate method support structuring compositional time Petri nets need to be found, it will take advantage of these recent developments in order to demonstrate that time Petri nets also can be successfully used in the specification and verification because it is allowed to be simplified into smaller subnet systems.

- *Discuss* the Hierarchical/Abstract theory has on time Petri nets with an overview of relevant concepts from compositional theory. The objective of this research section is to describe or construct TPNs for large systems by using existing elements and therefore reduce its design time.

Another aim of this project is to find or develop a suitable graphical tool with functions of time interval editing, real time model checking, a crucial aspect, this tool must support for hierarchical composition time Petri nets representation with the ability to design both top-down and bottom-up net and navigate around a hierarchical Petri net. There are many tools like INA, Predator, ProD, Visual Object Net ++, Tina [LINK], GraphLab [BH04] and so on, they provide functions varies from the most basic of graphical or textual editor, token game animation to hierarchy structuring, model checking, sophisticated properties analysis and state spaces analysis. But, among these Petri nets tools, do any one or more be introduced to implement our project's design and provide multilayered analysis required? If yes, using it in our project, otherwise, a new tool should be developed. This part includes tool development and implementation:

- *Design and program* a time Petri net editor tool that will make up the current lack of support for hierarchical composition time Petri nets and the limitations of analysis features in current editors. Support for hierarchical composition time Petri nets requires a suitable graphical representation and the ability to navigate around the complex time Petri nets system.

- *Implement* time Petri nets into the Graphical User Interface. Whiling implementation, the relationship between Petri nets components will be demonstrated. Important values like tokens, transition, firing time delay and connect points between different levels will be identified.

- *Calculate* State Class (SC) and *generate* the State Class Graph (SCG) ([BD91] [BH04] [BH05]) based on Petri nets tool selected or new programmed. Show how the state classes of higher-level components can be built from the state classes of its sub-components.

- *Execute* time Petri nets examples by using hierarchical composition theory and editing them on Petri nets tool selected or developed, both theoretical (Dining Philosophers) and practical (Flexible Manufacturing Systems) cases will show advantages of hierarchical composition TPNs when designing complex systems. These examples will help end users understand why complex systems can be modeled easier.

1.3 Synopsis

The organization of this thesis is as follows:

Chapter 2 starts by providing basic definitions, model checking conceptions, properties analysis and techniques for time Petri nets as a preliminary section. Additional important contents in this chapter are reviews and comparisons of historical and concurrent compositional techniques in the related field. **Chapter 3** considers hierarchical composition technique as the problem solving for complex systems design and modeling followed by previous reviews. **In Chapter 4**, features offered by existing Petri nets editors are also investigated and begin to design Petri net editor issues and program based on tool selected or new programmed. Coding graphical user interface and implementing hierarchical composition time Petri nets are also included in this chapter. This chapter also show how to design and implement via a theoretical (Dining Philosophers) example. **Chapter 5** studies a complex industrial-sized case of Flexible Manufacturing Systems (FMS). State Class Graph will be generated in each of cases and result will be analyzed. **Chapter 6** concludes this thesis, summarizing the research contributions, limitations, and presenting intention for the future work.

CHAPTER 2

PRELIMINARY KNOWLEDGEMENTS

In this chapter, basic notations and definitions of Petri Nets (PNs) and analysis techniques are briefly summarised for system modeling, then, we introduce the time extension in Petri Nets (TPNs) for specifying the behaviors of real-time reactive systems. The second and important part of the chapter is reviewing existing compositionality literatures and commenting their advantages and limitations. The role of comparisons among these related techniques is to help find an appropriate and effective approach for complex time Petri nets modeling.

2.1 Basics of Petri nets (PNs) and Time Petri nets (TPNs)

Since the area of Petri nets was initiated by C.A.Petri in 1960s, it has been developed tremendously in both the theory and the applications. Petri net model is one of the most powerful for modeling, simulating, and analyzing dynamic systems, although many other models of concurrent and distributed systems have been developed in these decades.

2.1.1 Petri nets (PNs) Definition

Petri nets (PNs) can be identified using graph based mathematical formalism for systems description. The graph for PNs always is divided into three types of objects: a set of places, a set of transitions, and directed arcs connecting places to transitions (input arcs) and transitions to places (output arcs). Places represent conditions, queue, buffer and other resources, while transitions represent events such

as send or receive. Graphically, transitions are depicted by bars of boxes, and places are depicted by circles. Use this elementary net can represent various aspects of modeled system. In addition, tokens often are asked to add to places for studying systems' states and their changes.

2.1.2 Time Petri nets (TPNs) Conception

The classical PNs allows for the modeling of states, events, conditions, synchronization, parallelism, choice, and iteration. Actually, basic PNs is limitative to describing real processes tend to be complex and extremely large. Furthermore, classical PNs do not allow for the modeling of data and time by reason lack of a temporal description and, therefore, fail to represent any timing constraints for time-dependent systems. To solve these problems, some elements have to be added the expressiveness of classical PNs, and some way to express computation on them. Such high-level Petri nets like: Extension with color to model data using Colored Petri nets; Extension with hierarchy to structure large models; Extension with time to deal with timing issues.

Time extensions were introduced to take also temporal specifications into account when using Petri Nets. Three are three main time extensions of Petri Nets:

- Timed Petri nets [RA74]: where transitions were associated with deterministic time durations
- Merlin's Time Petri nets (TPNs) [ME74]: where transitions have minimum and maximum time delays for firing

In Timed Petri Nets, transitions are fired as soon as possible, while the transition can be fired within a given interval for TPN. Among these models, TPNs is most widely used for real-time system specification and verification ([BV95]

[WD99]). In this chapter, only introduce Merlin's Time Petri Nets (Figure 2.1) in more detail

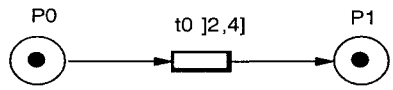


Figure 2.1 Merlin's Time Petri nets

• Definition of Merlin's Time Petri Nets (TPNs):

Like an example in figure 2.1, a TPN can be formally defined as 6-tuple TPN=$< P,T,I,O,M_0,SI >$ where:

– (P,T,I,O,M_0) is a classical Petri nets.

- $P=\{p_1,p_2,...,p_m\}$ is a finite set of places;
- $T=\{t_1,t_2,...,t_n\}$ is a finite set of transitions, $P\cup T\neq\varnothing$, and $P\cap T=\varnothing$;
- $I:(P\times T)\rightarrow N$ is the set of input arcs that places to transitions, where N is a nonnegative integers;
- $O:(P\times T)\rightarrow N$ is the set of output arcs that from transitions to places, where N is a nonnegative integers;
- $M_0:P\rightarrow N$ is the initial marking;

– SI is a static interval mapping. $SI:T\rightarrow Q^+\times(Q^+\cup\infty)$, where Q is the set of positive rational numbers; Time Petri Nets (TPNs) associate a finite enabling duration $SI(t)=(\alpha^s,\beta^s)$. (α^1,β^1) is the static firing interval of transition t in a net, where α^s is called the static earliest firing time, β^s is called the static latest firing time.

• Transition Enabling Rule:

A transition t is said to be *enabled* if each input place p of t contains at least the number of tokens equal to the weight of the directed arc connecting p to t. Mathematically define it: $\forall p \in P,\ M(p) \geq I(t,p)$.

• **Firing Rule:**

Firing t at the market M yields a new marking M'. A firing of an enabled transition t removes tokens from each input place p and creates new tokens to the output place p. Mathematically define it: $\forall p \in P, M'(p) = M(p) - I(t,p) + O(t,p)$.

For a transition to occur it must have been continuously enabled for at least the earliest firing time and for not more than the latest firing time. The state of a TPN can change due to two reasons:

- Time progression (bounded by the latest firing times of enabled transitions) does not change the marking (remove tokens from Input places to Output places) but updates the clock values for all transitions. The maximum time than can pass at any state is bounded by the latest firing times of enabled transitions.

- Time transition fires which change the marking with *no time* and update the *SI(t)* function excluding disable transitions and adding newly enabled ones.

2.2 Important Properties of TPNs

TPNs nets exhibit properties, which can be verified. *Liveness, Safeness* and *Boundedness* are three important properties of Petri nets. Before making these properties understandable, we first give definitions of *Reachable Marking*. Properties of Petri Nets depend on the initial marking (M_0).

• Reachable Markings:

A marking M is reachable if and only if there exits a sequence s that leads to a marking of PN can be obtained from the initial marking M_0. If one or more nodes of in Reachability Graph (*RG*) have not output arcs, this PN system is said to contain deadlock. Mathematically, $\overrightarrow{M_0} = \{M, \exists, M_0 \xrightarrow{s} M\}$

• Live and Liveness:

Mathematically, $\forall M \in \overline{M_0}$, $\exists_s, M_0 \xrightarrow{s,t}$. A transition t is said to be live if and only if for each marking M reachable form the initial marking M_0 (whatever state or marking the system is in) there exists a marking M'. A PNs is said to be Lk-live, for marking M_0, if every transition t in the nets is Lk-live, k = 0, 1, 2, 3, 4. Live is equivalent to deadlock-free. Live is desirable in most systems because deadlock is avoided and the complete system is always accessible.

• K-Bounded and Boundedness:

Mathematically, $\exists k \in N, \forall p \in P, \ \forall M \in \overline{M_0}, \ M(p) \le k$. A place is said to be k-bounded in a PN system, if and only if there exists a $k \in N$ such that for all markings $M(p) \le k$. Further a Petri net is k-bounded if in all possible markings, and for all places, the number of tokens on a place is not greater than k. Boundedness determines that in all markings each place contains a finite number of tokens. Boundedness is important to prevent buffer overflows in systems.

• Safeness:

Place is safe if in all possible markings, the number of tokens on the place is never greater than one (1-bounded). In addition, a Petri net is safe if all places in

the net are safe. Safeness is of importance for places that represent conditions.

2.3 TPNs Analysis Summary

Fundamental and most widely applied method for analyzing TPNs ranges from the firing sequences' simulation to complex performance analysis. This section will describe briefly some of the more common analyzing techniques such as *Invariant Analysis, Reachability Analysis, State enumeration, Performance evaluation, Token Game Simulation.* In this project, we emphasize introduction on below three analyses:

• Reachability Analysis:

Reachability analysis ([BD91] [BM83]) generates the reachability tree (show in Figure2.2) of a TPN. It starts at the initial marking, with a new marking created for each transition. This is repeated for all new markings until all possible markings have been covered. It permits the automatic translation of behavioral specification models into a state transition graph made up of a set of states, a set of actions, and a succession relation associating states through actions [BV95]. This representation makes explicit such properties as deadlock and reachability ([ZH95] [WD00]), and allows the automatic verification of ordering relationships among task execution times [TY95].

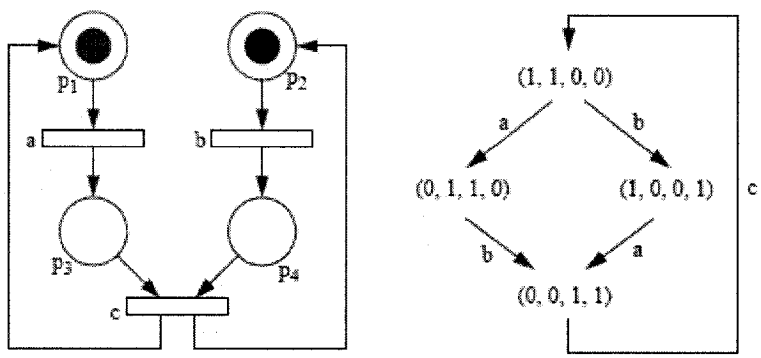


Figure 2.2 A Petri Net and its Reachability tree

• Token Game Simulation:

Simulation provides the simplest analysis of TPNs by firing all enabled transitions step by step from its initial state. User can choose the transitions to fire and observe the net state. When a transition is fired, tokens are moved accordingly and the process is repeated, another enabled transition can be selected. This animation approach can be an effective way to view transition firing sequences, but does not provide any more information in this project.

• State Enumeration:

State Enumeration analysis is based on building a graph which represents all the states reachable in a Petri net by firing its transitions. For TPNs, states are grouped in state classes (detail introduction is presented in the next part) with common features. Each state class is composed by a marking and a firing domain (enabled transitions set, each transition with its respective dynamic firing interval).

As an analysis module to perform, *State Enumeration* is important analysis technique in this project and will be described in more detail in the following section.

2.4 State Classes Method for Analyzing TPNs

In TPNs, timing constraints are expressed in terms of minimum and maximum amount of time elapsed between the enabling and the execution of each action. This allows a compact representation of the state space and an explicit modeling of time-dependent systems. An enumerative method proposed by Berthomieu and Diaz in ([BD91]), which supports computing the state space of TPNs models. In this part, based on the concept of *state, state class* and *state class graph*, this enumerative

technique for TPNs are introduced.

2.4.1 State

By Berthomieu and Diaz, the notion "state" is used for describing many situations of the Time Petri nets and it is defined by a marking and time intervals for the enabled transitions. A state S of TPNs can be defined as pair $S = (M, I)$, where M is a marking and I is a set with entries of inequalities, each entry describes the upper and lower bound of the firing time of an enabled transition. Different states may have different numbers of entries.

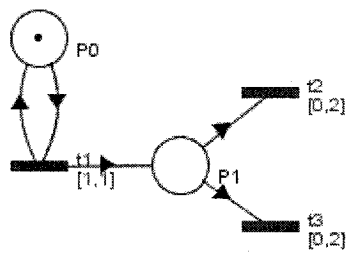


Figure 2.3 A simple Time Petri nets model

Consider the TPNs model given in Figure 2.3. Its initial state is $S_0 = (M_0, I_0)$, where: $M_0 = (1,2,0,0,0)^T$ and $I_0 = \{4 \le \theta(t_1) \le 9\}$. Here, using $\theta(t_i)$ to represent the allowable firing time of transition t_i enabled in a given marking.

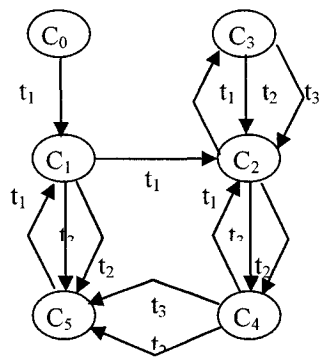
2.4.2 State Class (SC)

State class $C = (M, D)$ of a TPN is a pair, associated with a firing sequence ω from the initial state S_0, consisting of:

- M is a marking of the class: all states is the class have the same marking.
- D is a firing domain of the class; it is defined as the union of the firing domain of all states in the class.

2.4.3 State Class Graph (SCG)

A state class graph is the graph of all state classes. For example, considering the TPNs model given in Figure 2.3, we attain state classes and reachability graph (state class graph) as below (Figure 2.4):



(Initial class)C_0 $M_0=p_0$; $D_0=\{1\le\theta(t_1)\le1\}$	C_1 $M_1=p_0,p_1$; $D_1=\{1\le\theta(t_1)\le1,$ $0\le\theta(t_2)\le2,$ $0\le\theta(t_3)\le2\}$	C_2 $M_2=p_0,p_1(2)$; $D_2=\{1\le\theta(t_1)\le1,$ $0\le\theta(t_2^0)\le1,$ $0\le\theta(t_2^1)\le2,$ $0\le\theta(t_3^0)\le1,$ $0\le\theta(t_3^1)\le2\}$
C_3 $M_3=p_0, p_1(3)$; $D_3=\{1\le\theta(t_1)\le1,$ $0\le\theta(t_2^0)\le0,$ $0\le\theta(t_2^1)\le1,$ $0\le\theta(t_2^2)\le2,$ $0\le\theta(t_3^0)\le0,$ $0\le\theta(t_3^1)\le1,$ $0\le\theta(t_3^2)\le2\}$	C_4 $M_4=p_0, p_1$; $D_4=\{1\le\theta(t_1)\le1,$ $0\le\theta(t_2)\le2,$ $0\le\theta(t_3)\le2,$ $\theta(t_2)-\theta(t_1)\le1,$ $\theta(t_3)-\theta(t_1)\le1\}$	C_5 $M_5=p_0$; $D_5=\{1\le\theta(t_1)\le1\}$

Figure 2.4 State Class Graph of TPNs in Figure 2.3

2.5 Temporal Logics Model Checking

Temporal logic model checking (Figure 2.5) is an automatic and formal technique that seeks to establish a mathematical proof that a TPNs works correctly, this formal method provides: *Modeling Language* for describing the system; *Specification Language* for describing the correctness requirements; *Analysis Technique* for verifying that the system meets the correctness requirements.

Temporal logic specification:

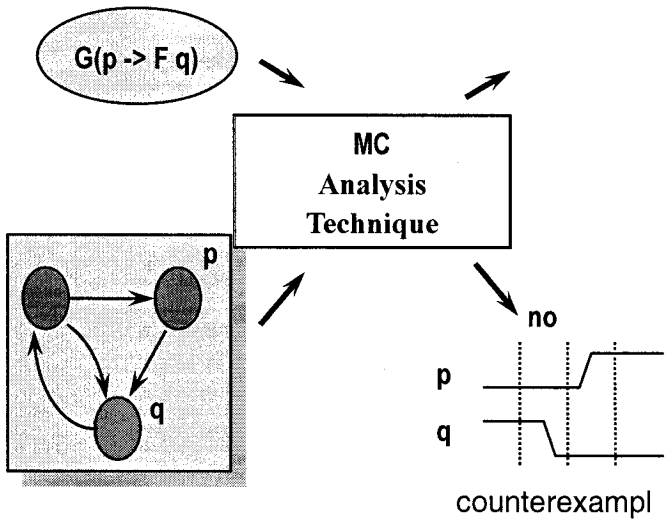


Figure 2.5 Model Checking Principle

Model-Checking models a finite-state system using a variety of modeling languages, and verify it using a variety of checkers for deadlock, reachability and model for the temporal logics of TPNs. To check a temporal property, we may formalize it in specification language *Computation Tree logic* (CTL) and *Linear Temporal Logic* (LTL). They differ in how they handle branching in the underlying computation tree:

2.5.1 Linear Temporal Logic (LTL)

In LTL operators are intended to describe properties of all possible computation paths. LTL is built up from proposition variables p_1, p_2,..., the usual logic connectives *conjunction* ($\wedge$), *disjunction* ($\vee$), *negation* ($\neg$), *implication*($\Rightarrow$), *equivalence* ($\Leftrightarrow$) and the temporal operators *eventually* (F), *always* (G), *next* (X) and *until* (U). LTL formulas are generally evaluated over paths and a position on that path. A LTL formula as such is satisfied if and only if it is satisfied for position 0 on that path. In LTL for TPNs [TM04], each point of time only has one possible future as opposed to branching time logics which are interpreted on the reachability graph. Hence, for a TPNs system to satisfy a LTL formula ϕ all its executions must satisfy the formula.

2.5.2 Computation Tree logic (CTL)

In CTL [YR98] temporal operators it is possible to quantify over the paths departing from a given state. It is often used to express properties of a system in the context of formal verification or model checking. It uses atomic propositions as its building blocks to make statements about the states of a system. CTL then combines theses propositions into formulas using *logical operators* and *temporal operators*:

- *Logical Operators*: The logical operators are the usual ones: the usual logic connectives *conjunction* ($\wedge$), *disjunction* ($\vee$), *negation* ($\neg$), *implication* ($\Rightarrow$), *equivalence* ($\Leftrightarrow$). Along with these operators CTL formulas can also make use of Boolean constants true and false.
- *Temporal Operators*: *All paths* (A), *some paths* (E), *eventually* (F), *always* (G), *next* (X) and *until* (U). In CTL, temporal operators only use AX and EX, AG and EG, AF and EF, AU and EU.

2.6 State Classes Preserving LTL or CTL

State classes were proposed for finitely representing the state graphs of bounded Time Petri nets, preserving markings and complete traces. Distinguished from other complete traces, in works of [BV03] and [BH05], state classes will referred to as state classes preserving linear temporal logic.

The set of linear state classes of a Time Petri net is finite if and only if the Time Petri net is bounded [BM83]. Property is undecidable but there are decidable sufficient conditions for it [BV03]. Boundedness is a characteristic in state class graph generation. Time Petri nets considered often are assumed bounded in this case. In terms of temporal logics, state classes preserving LTL formula (*LSCG*) are complete trace equivalent. State classes preserving LTL hold the same markings and firing sequences with ordinary State space. Preliminary implementations of the algorithms for building strong and atomic state class graphs have been integrated in a tool name *Tina*. Beside these constructions, *Tina* offers LSCG (state class graphs preserving with LTL properties).

The work in [BH05] deals with the verification of CTL* properties of Time Petri Nets (TPN model) and contracts the generally infinite state space of the TPN model into a finite graph that preserves its CTL* properties. Such state class graph can be constructed using a refinement technique. This approach ([BH05]) is implemented in *GraphLab* tool. Experiments results have shown that the contractions are very appropriate to boost the refinement procedure and reduce computation times by factors reaching four and more in certain cases. The most important, state class graphs have also been reduced in size.

CHAPTER 3

COMPOSITION CLASSIFICATION AND COMPARISON

First purpose of this chapter is studying basic knowledge of compositionality; The second step, reviewing various composition techniques literatures and to propose classification aspects for the great variety of composition for Petri nets; Discussing applicability and application fields of primary composition techniques; And finally, selecting one effective composition to use for complex Time Petri nets modeling.

3.1 Introduction of Compositional Time Petri Nets

It is an old idea to handle the complexity of systems by composing them out of smaller components. Although Time Petri nets (TPNs) have for several years been applied as a formal model in which one could describe and analyze systems, it can be argued that they are not widely used in industrial-sized systems. The main reason for this can be attributed to a lack of compositionality which means that TPNs were unable to deal with large complex or even midsized systems. In this kind of systems, state-explosion is a commonly problem that it is difficult to enumerate all reachable states. Compositionality which is composed of components and connectors is so crucial for practical application of TPNs systems. In the past few years, some attempts to bring compositionality and modularity into Petri nets ([RS83][SM83][BV95][RA96][ES96][JA98][CO95]) did result in a substantial progress in system modeling. This project will take advantage of these developments

in order to demonstrate that composition TPNs can be successfully used in the construction and specification of large systems. We also outline such a framework can support modular verification techniques. However, their detail discussion lies outside the scope of this project.

Due to there is little and sparse attention of an overview of available composition mechanisms for large Petri nets models with time extended, this project first will give precise notion of composition and investigate, review existing approaches in compositionality area of Petri nets, offer an up-to-date detailed survey of some of these approaches. As a result, an effective compositional approach will be presented to construct individual Petri net components, which can be combined into larger components in a uniform fashion while at the same time facilitating compositional analysis.

3.2 Elementary Conceptions

Numerous approaches have been proposed that try to manage model size through the introduction of structuring mechanisms allowing model compositionality. It is necessary and obvious requirement for analysis the construction of any sizable model. Before dressing the practical application selecting in this project, Compositional Concepts of *components,* we should first informally describe these conceptions *Connector, Component (Subnet, Supernet), Fusion, Folding, Abstract / Atomic, Hierarchy/Refinement, Algebra, Object-oriented.*

• ***Components (or Subnets) and Connectors:***

Components can be called *subnets* with inputs and outputs instead of trying to directly refine either places or transitions [JA98]. *Connectors* are a simple place or transition in TPNs to describe the interaction among components. Each component

has external connectors and internal structure. Figure 3.1 shows the example of a component that has one input and one output connector which connect to external structures.

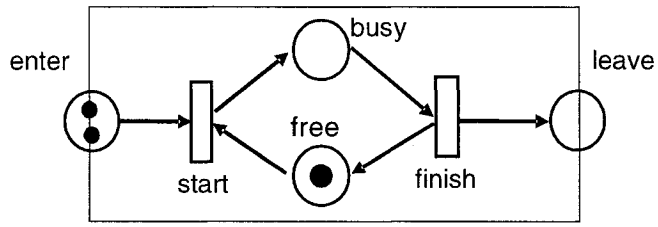


Figure 3.1 Example of Component and Connectors (enter, leave)

Composite components can be defined assemblies of sub-components; whereas *elementary components* can have Petri-Net specification. Modern complex time-dependent systems often use components (C1,C2,C3,...) interacted with each other through connectors (port1, port2, port3,...) to construct compositional TPNs (see Figure 3.2). Components will be considered to construct inputs and outputs subnets of TPNs, which can be embedded into other components [JA98].

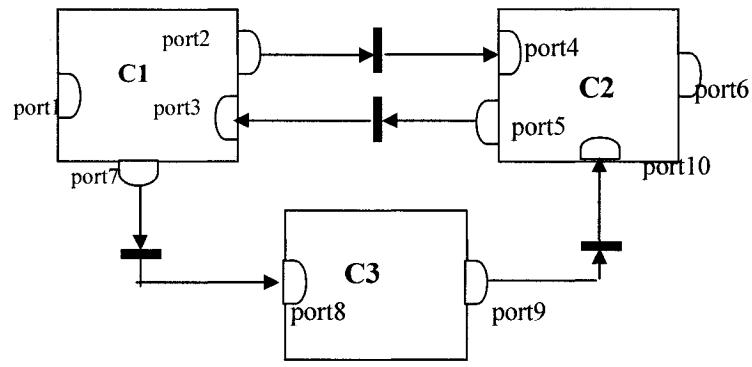


Figure 3.2 Components Interacted with Each other through Connectors

• ***Subnet and Super-net:***

When a refinement adds an encapsulated net to an existing net, the former is

named a **subnet**, whereas the latter is named a **supernet**. One of its elements becomes a representation for the subnet. An abstraction is the reverse process: a subnet is replaced by a single node.

• ***Abstract and Atomic:***

If the element in a Petri nets can be replaced by a subnet (or other elements) which represents its behavior in more detail, we call it is *abstract* element. Otherwise the class is called *atomic* element. *Abstractions* allow designer to ignore (or hide) implementation details of structures for large systems.

• ***Hierarchical Abstraction / Hierarchical Refinement:***

Complexity model system can be handled by hierarchical structuring mechanisms [VA79]. Model structuring mechanisms of adding hierarchy to Petri nets is the abstraction (or refinement) of either or both of its standard atomic places and transitions. Compositional PNs representation is considered such that in a top–down refinement step:

- Start with high-level abstraction (Means that at a higher level the state of a subnet must be represented in less detail than at a lower level).
- Progressively *refine*.

Hierarchical abstraction/refinement technique supports the definition of components and their reusability. It takes advantage of graphical expressiveness capabilities in order to hide/show details of the model in a consistent manner.

• ***Fusion:***

Fusion is a kind of composition. In order to avoid long arcs connections which will bring to legibility problem in PNs structure formalism, places or transitions

fusion are used as a drawing convenience to fold a set of elements(places or transitions) into a single one, Afterwards, fusion took its place also as a fundamental concept to support model compositionality. For example (see Figure 3.3): A fusion place is a place that has been equated with one or more other places, so that the fused places act as a single place with the same type and number of tokens [SA97]. The fusion place capability allows places in a Petri Net that exist in different locations in the network to act functionally as if they were the same place. Such places are called fusion place.

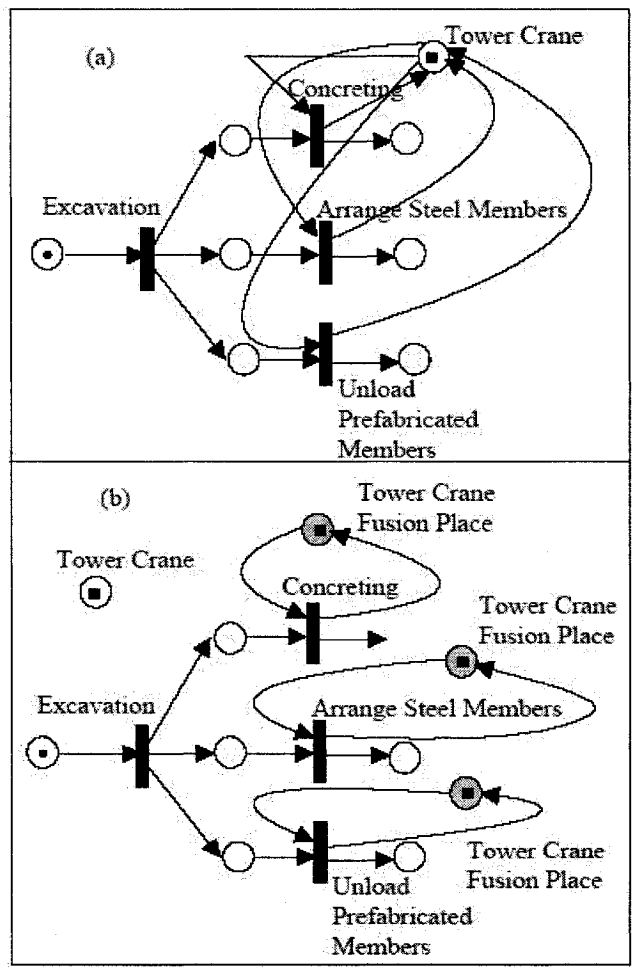


Figure 3.3 Petri Net Modeling using Fusion Places Example

• ***Folding:***

Another kind of composition is folding abstraction. Folding means "mapping many elements to one element". There are two folding composition types:

- ***Token based folding*** (The folding is accomplished through the use of tokens as data structures)
- **Element based folding** (The elements to be merged can be places, transitions and abstract elements, the folding is accomplished by element fusion). As for abstract elements, it will be presented in the following sections, within refinement/abstraction structuring mechanisms.

• ***Object-orientation:***

Object-oriented concepts include inheritance and the associated polymorphism and dynamic binding. Compositionality using Object Orientation merging Petri Nets [BO01] combines the maintainability and reusability of Object Orientation (OO) and the advantages of Petri Nets a graphical interface and a sound theoretical background. In Petri nets, not only token data but also the Petri nets itself can be object-oriented. When the object oriented approach taken, every object is an instance of a particular class. The inclusion of object oriented concepts provides facilities for creating components that encapsulate behavior and state, and thereby supports: abstraction, refinement, encapsulation, and reuse and sharing.

• ***Compositionality:***

According to above definitions, formal compositionality concept means the interconnection of several Petri net models. It is based on terms of *architecture, elements (places, transitions and components),* composition operators (i.e., abstraction, hierarchy) and *system.* Architecture determines how elements can be combined to form new elements. An element that is usable for a certain purpose. When defining the compositionality of the architecture with respect to at least one

element properties like reachability, safety. Compositionality can re-use the information about the subnets of a component to obtain information.

3.3 Compositional Mechanisms Classification

Various approaches have been proposed that try to manage model size through the introduction of structuring mechanisms allowing compositional model representations. This section proposes a classification for Petri nets' structuring mechanisms and discusses each one of them. These mechanisms include node (place or transition) fusion, node (place or transition) sharing, object-oriented inspired Petri nets, and abstraction/refinement, algebra, workflow, trace theory, and so on. One or more representative papers are used emphasizing the application of the presented mechanisms to specific areas, namely to systems modeling, and software engineering in time-extension system or normal system, where one kind of mechanisms modeling plays a major role. There are many approaches for structuring compositional Petri nets, according to literatures review, we classify these mechanism like below (see Figure 3.4)

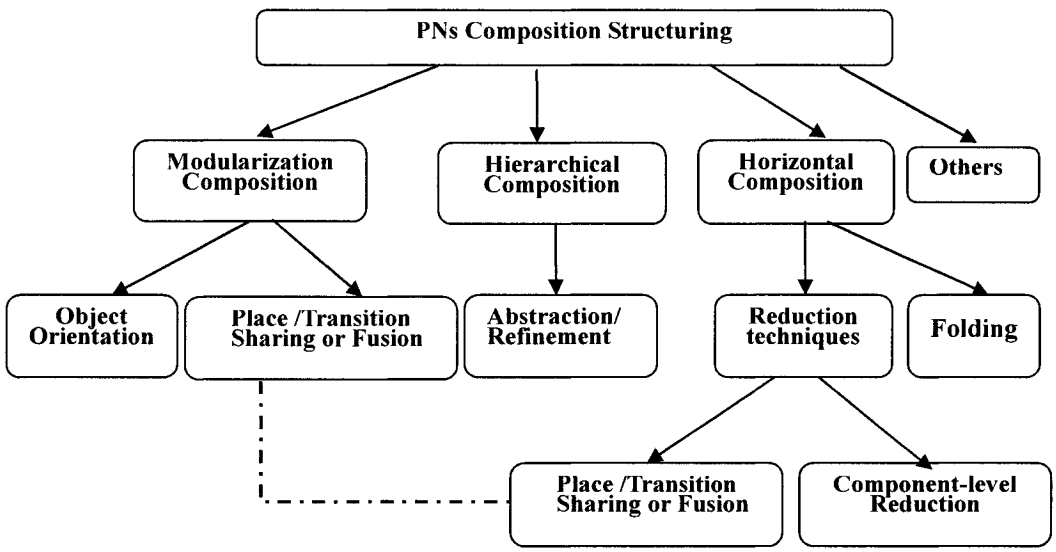


Figure 3.4 Compositional Petri nets Structuring Mechanisms Classification.

Besides above primary structuring methods, there are still some composition methods like *algebra* (pBC, M-nets, Well-formed net), *work-flow, trace theory*, which will be introduced summarily in section 3.5.3 of this Chapter.

3.4 Satisfactions Requirements

When a compositional structuring is done, it must satisfy the following requirements:

- ***Behavior and properties equivalences:***

  Components may be replaced by smaller one, elements (places /transitions/ PNs-itself) may be reduced by folding, etc. Whatever methods use, the equivalent compositional nets will be got without affecting significant properties of the whole model.

- ***Whole net likeness:***

  No matter horizontal, hierarchical or modularization technique using for compositionality. It is clear that the further system specification is done in a way that all elements (places, transactions, arcs, components and connectors) that run on the same level. Such a composed system specification may look like the whole net (shown by Figure 3.5) [RW02].

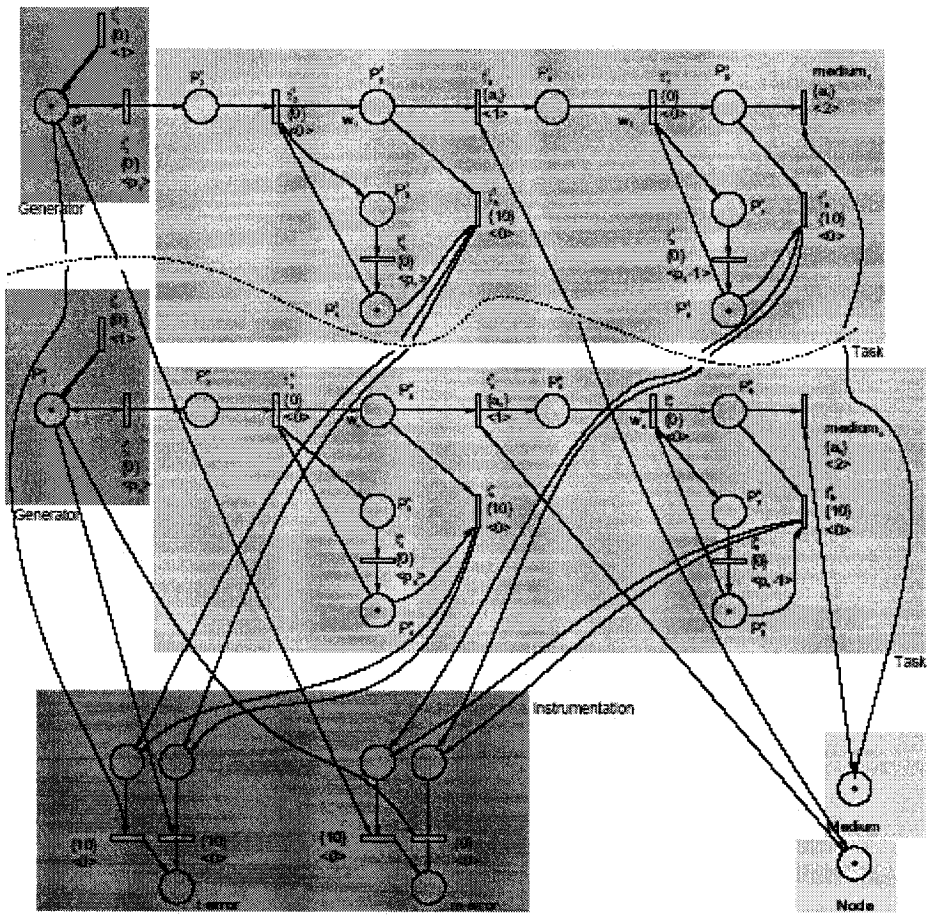


Figure 3.5 Whole net likeness after Compositional Structuring

3.5 Reviews, Comparisons and Comments

In this section, some typical existing works in the area of compositionality into the PNs formalism and the criteria for the construction and analysis of systems models are introduced as follows.

3.5.1 Reduction Technique for Compositionality

The goal of net reduction is to increase the effectiveness of TPNs analysis, like

all reachability based methods, suffers from the state explosion problem. Petri net reduction is one key method for combating this problem. Several authors have investigated reduction techniques for Petri nets. In particular, Berthelot originally developed a large set of reduction rules for general Petri nets [BE83]. Berthelot's approach was subsequently extended by the project in [SB96], it defines a set of reduction rules for TPNs. Finally, it proves that each reduction rule satisfies the definition of equivalence for reduced and unreduced nets. Reduction technique is applied by *places/transitions' transformation rules* and *component-level reduction.* We introduce these two approaches below concisely and give more emphases on the second one.

3.5.1.1 Transformation rules for TPNs

Reduction technique is usually proposed to transform the original model of the system into a simpler one. For example, in ([BA02] [SB96]), compositionality is realized by taking advantages of complexity reduction technique which provided by the application of *Interval arithmetic, Reduction rules and Nets equivalence:*

- ***Interval arithmetic:*** Let $I_1 = [a_1, b_1]$ and $I_2 = [a_2, b_2]$, with $0 \leq a_1 \leq b_1 \leq +\infty$ and $0 \leq a_2 \leq b_2 \leq +\infty$. Then the internals $I_1 + I_2$ can be defined as $[a_1+a_2, b_1+b_2]$ and $I_1 - I_2$ can be defined as $[a_1.a_2, b_1.b_2]$ if $a_2 \leq a_1$ and $b_2 \leq b_1$. For PNs with timed system, this technique consists of building the reduced model and its constraints graph representing a set of timing constraints. Preliminary experiments in their projects for the analysis of TPNs have shown that net reduction can be quite effective.

- ***Reduction rules:*** Include Serial fusion; Pre-fusion/Post-fusion; Lateral fusion; Redundant Places; Parallel Redundant Places; Removal of useless end places; Removal of empty begin places; Unnecessary marked begin places.

- ***Nets equivalence:*** Let U be the set of transitions of N which are left completely unmodified by transformations. The reduced net Nr is equivalent to the original one N after the considered transformation. The notion of equivalence concentrates on preservation among unmodified transitions and guarantees that crucial timing and concurrency properties are preserved, such as liveness.

An example is given. Components' composition is accomplished through transition fusion (Figure 3.6), three submodels (A, B, C) are components to be interconnected. Transition fusion is used to compose the three models into the global model.

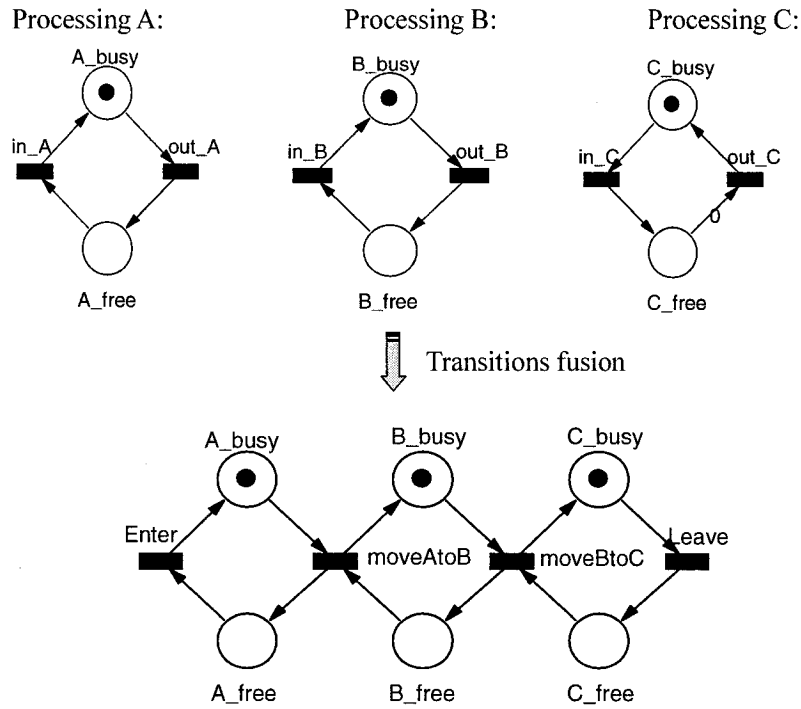


Figure 3.6 Transition fusion composing three models into the global model.

3.5.1.2 Component-Level Reductions for TPNs

• ***Introduction***

Reachability method is the fundamental and most widely way for analyzing TPNs. However, it is difficult to enumerate its reachable states because of the state-explosion problem. Compositional TPNs (CTPNs) model ([JC98], [WD00]) was introduced based on Sloan et al's approach [SB96] which developed several reduction rules for TPNs analysis to reduce the complexity of TPN analysis to some extent. More interesting is the technique described in [BV95] where a compositional approach to the analysis of time Petri nets components is proposed.

CTPNs models ([JC98], [WD00]) is a modularized time Petri net (TPN), which is composed of *components* and *inter-component connectors (connections in brief)*. It also proposes a set of component-level reduction rules for TPN's. The operational component models describe the real-time behavior and communication interface of the component or subsystems; the connections specify how the components interact with each other. All connections are defined using only communication interfaces to change the design of the individual components flexibility.

Communication of components is serviced by ports which provide the linkage between the operational design (components and connections) and the descriptive architectural constraints. All constraints are specified by using ports only. The set of constraints can be logically divided into three classes: *component constraints, environment constraints, path constraints.* A component constraint describes the timing properties of a component that its environment expects from it, an environmental constraint describes the timing properties that a component expects from its environment, and a path constraint describes the timing constraint for message transmission among components.

• ***Definition and Component-level reduction rules***

In order to give the example of CTPNs which consists of components and connections elements in the next paragraph, now we turn to give the related definitions.

$TPN = (P, T, I, O, M_0, SI)$ is the TPNs model of a component,

$$PORT\_IN = \{p \mid p \in P, {}^*p \cap T = \varnothing\};$$

$$PORT\_OUT = \{p \mid p \in P, p^* \cap T = \varnothing\};$$

$$PORT = PORT\_IN \cup PORT\_OUT = \{p \mid p \in P, {}^*p \cap T = \varnothing \vee p^* \cap T = \varnothing\}$$

Then, $\forall p \in PORT\_IN$ is called an input port of the component, $\forall p \in PORT\_OUT$ called an output port of the component, and $\forall p \in PORT$ called a port of the component. The component TPNs model of a component C is a 2-tuples, $Nc = (TPN, PORT)$, Where: TPN is the time Petri net model of the component and $PORT \subset TPN.P$ is the set of ports of the component. Let N be the CTPNs model of a real-time system and let

$$N.PORT = \{p \mid p \in P, {}^*p \cap N.T = \varnothing \vee p^* \cap N.T = \varnothing\},$$

Where: N.T is the transition set of N. $\forall p \in N.PORT$ is called the external port of the modeled system. $\forall p \in \bigcup_{i=1}^{k} NCi.PORT - N.PORT$ is called the internal ports of the modeled system. Every internal port of a CTPNs model must be connected at least one connection transition.

For the fundamentals of analyzing Component Time Petri Nets (CTPNs) models, we use Component-level reduction rules Figure 3.7 for TPNs [JC98] preserve the external observable timing properties of the components to be reduced. Each of these Component-level reduction rules transforms a TPN component to a

very simple one while maintains the net's external observable timing properties. Consequently, the proposed method works at a coarse level rather than at an individual transition level. Therefore, one requires significantly fewer applications to reduce the size of the TPN under analysis than those existing ones for TPNs.

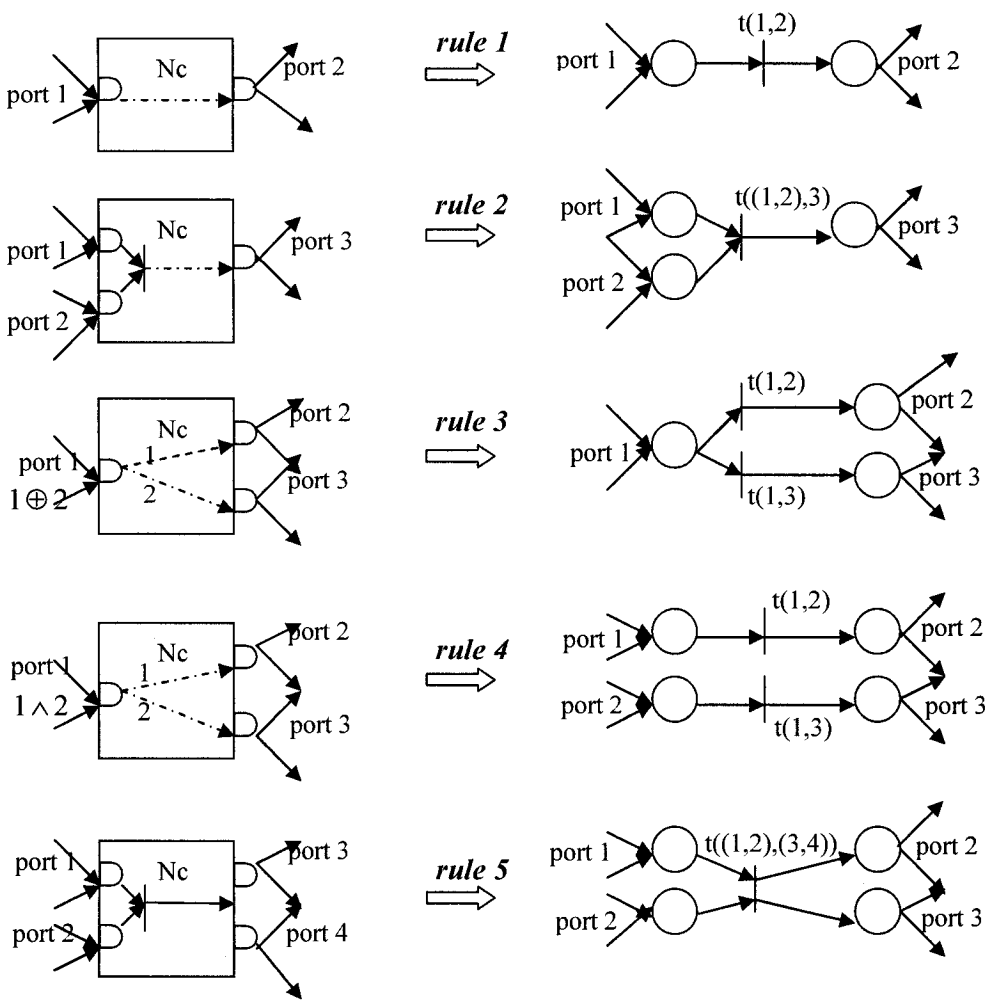


Figure 3.7 Illustration of Component Reduction Rules

• ***Examples of CTPNs Structure and Component Reduction Rules***

As an example of CTPN, Figure 3.8 has three components C1, C2 and C3 for the high-level model. Component C3 is further refined at the low level design into the composition of components C31, C32, and C33. The model at any level must

satisfy timing constraints specified at that level. Each component model has two parts:

1) Communication ports (denoted graphically by half circles), including input ports (e.g., port 7) and output ports (e.g., port 6).
2) A TPN that describes the time-dependent operational behavior of the component and it defines the semantics associated with the ports.

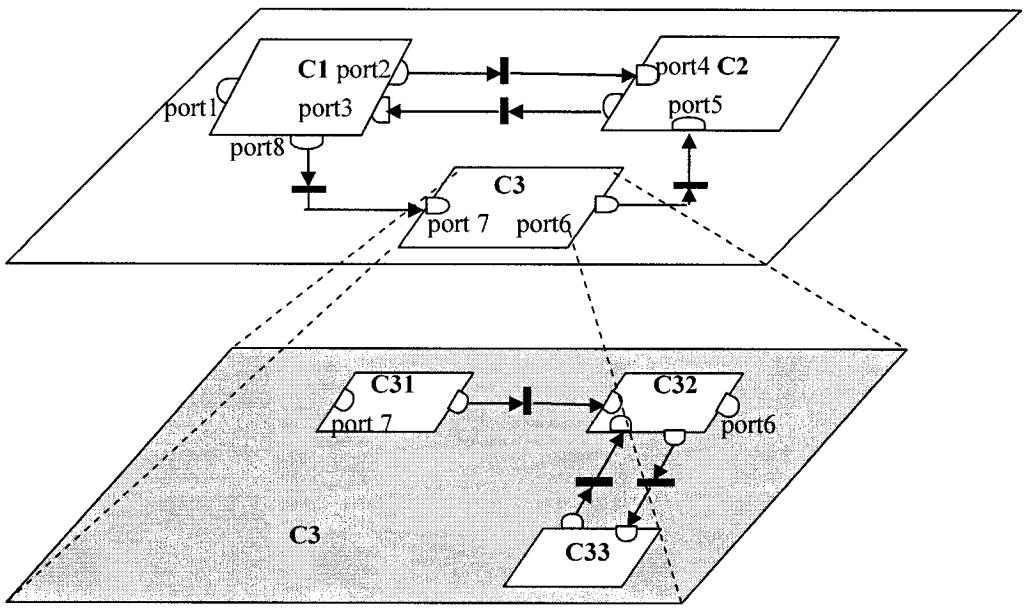


Figure 3.8 Framework of Compositional Time Petri Nets structure.

At the high-level of this CTPNs example, component C1, C2 and C3 construct a cyclic *component environment (CE)*: C1 and C2 are *CE* of C3. At the low-level, C32 can be said as C31 and C33's *CE*, and the *CE* of C31 is C32. At any given level, ports can also be divided into *external ports* (e.g. port 1 in the high level, port 6 and port 7 in the low level model) and internal ports (e.g. port 2 and port 4). Internal ports are defined to communicate with other components, while external ports are used to describe the inputs and outputs from and to the environment of the system.

The communication between a component and its environment is solely through the ports. A connector represents a channel of interaction between components. It is modeled by a simple TPN and defines the direction of message flow and delay in the channel. For example, components C1 and C2 have a request–reply relationship that is modeled by a bidirectional channel. The communication is asynchronous message passing. That is, the sending component does not suspend itself if there is a concurrent activity available.

3.5.1.3 Advantages and Limitations

Referred to as a state-explosion problem, CTPNs propose a set of component-level reduction rules to reduce the verification complexity. Each of the reduction rules transforms a TPN component to a small one while maintaining the net's external observable timing properties. In generally, the process of verification is driven by showing that the components and their composition satisfy their corresponding constraints at every design level. CTPNs [JC98] cut down the verification complexity by supporting horizontal as well as vertical composition-ability. At any given level, verification proceeds by analyzing the components against their corresponding component constraints one at a time and then composing these results to deduce system-side properties as specified by connection and cross-connection constraints at that level. Thus, the complexity of analysis is proportional to the size and number of components, rather than the size of the entire model. After a design has been verified at a given level, it is further refined into lower level design. The high level constraints are propagated to the lower level. Then the design at the lower level is checked against the lower level constraints, and the refinement process continues. In conclusion, the use and benefits of these reduction rules are illustrated by reducing dramatically the size and the verification complexity in regular Time Petri Nets,

Limitations of CTPNs: While mention of advantages of using CTPN, we need to know limitation of this approach is that we must assume that dependency graph does not contain a cycle firstly. In CTPNs, when the component Ci depends on the components Cj, denoted by $Ci \rightarrow Cj$, then Cj has to be analyzed before Ci, if $Ci \rightarrow Cj$ and $Cj \rightarrow Cl$, then Cl has to be analyzed before Cj. So, it is usually assume that the dependency graph does not contain a cycle. In other words, the first component does not depend on any other components so it can be verified and reduced to a simple TPN. The second component depends only on the first one, which has already been reduced, and so on.

Limitations of Transformation: This transformation rules must check the existence of feasible controls. Only if the feasible controls exist, their associated control or the original model is made possible in a hierarchical way.

3.5.2 Modular Approach

A modular approach to Petri nets modeling makes larger systems easier to handle, and a large degree of reuse significantly decreases the development time and reduces maintenance cost. Place/Transition sharing and Object-Orientation are two important modular approaches for compositionality.

3.5.2.1 Places/Transitions Sharing and Fusion

• ***General Introduction and Basic Definitions***

Like transformation [BA02], *Modular Composition* ([CP00], [BV95]) is one of the primary approaches to the solution of the problem of TPNs, which is, it is very

hard to analyze due to the net and of its state space.

Modular compositional method can be performed by *place invariants* (It express invariant relations on the markings of places) and *state spaces* (This method's basic idea is to compute all reachable states and state changes of the system, and represent reachable states as directed graph). The two most important ways are always illustrated via shared place and shared transitions. Sharing is often accomplished using place fusion sets and transition fusion sets. So, there are two kinds of fusion sets as follows.

Place fusion sets: The sets of places to be fused together are called place fusion sets. All places in the fusion sets share the same markings: when a token is added to a place which belongs to a place fusion set, all places of the place fusion set will have the same token added. All places in a place fusion set have identical initial marking.

Transition fusion sets: Each transition of a transition fusion set describes a part of a more complex action and all parts must occur simultaneously, as one indivisible action. Transition fusion set is enabled if all the transitions in the fusion set are enabled. Another character is a transition can be a member of several transition fusion sets.

State space: Is known as *occurrence graph* or *reachability graph*. The main reason for working modular state spaces is to alleviate the *state space explosion problem*. For example, like Figure 3.9, Compositionality of ordinary PNs is realized by smaller the size of the occurrence graphs of the modules using transition sharing.

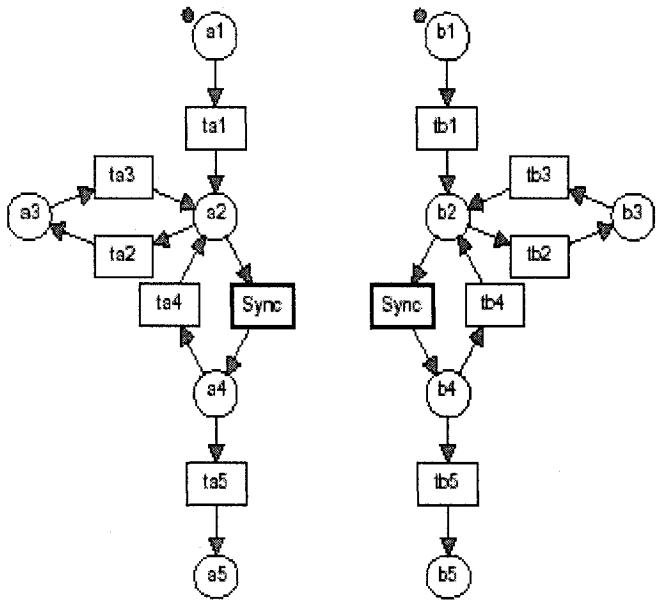


Figure 3.9 Two Modules (A and B) communicating processes example.

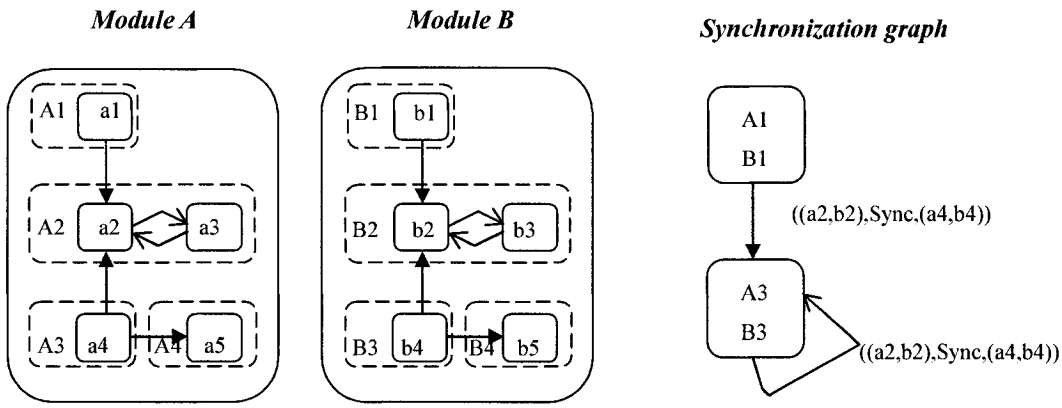


Figure 3.10 Modular state spaces example with transition sharing

A modular state space is composed of one local state space per module (is shown in Figure 3.9: Module A and Module B) and a synchronization graph (is shown in Figure 3.10) which captures the communications. A local state space only contains the local information. The synchronization graph provides information on the state space of the overall system and the occurrences of fused transitions. The

idea is to generate a state space for each module and the information necessary to capture the interaction between modules, and in this way avoid the construction of the full state space.

Another typical project using modular for compositionality in [BV95] constructs module to specify complex systems with the timing semantics of TPNs. In that paper, the state space of each individual module can be separately enumerated and assessed under the assumption of a partial specification of the intended module operation environment. State spaces of individual modules can be recursively integrated, to permit the assessment of module clusters and of the overall model, and to check the satisfaction of the assumptions made in the separate analysis of elementary component modules. The joint use of incremental enumeration and intermediate of local events allows for a flexible management of state explosion, and permits a scalable approach to the validation of complex systems.

3.5.2.2 Object Orientation Approach

• ***Introduction***

Among research works temp to realize compositionality, the Object Oriented Petri Nets (OOPNs) ([LA95][ES96][BO01]) theoretically formalization and graphical formality is an efficient way to implement. Incorporate Object-oriented structuring into Petri Nets will solve the weakness in Petri Net formalisms: The absence of compositionality has been one of the main critiques raised against PNs models. The goal of this formalism is more flexible and powerful structuring primitives and supporting for software simulation practically.

• ***Object-Oriented Conceptions***

In OOPNs formalism (see example in Figure 3.11), Object-Oriented

Conceptions (such as: Class, Object, Encapsulation, Instance, Inheritance) using by merging Petri nets.

- Basic Object-Oriented knowledge:

- ***Class:*** Is just a **template** for the **Object**
- ***Object:*** Is an **instance** of (belongs to) a **Class**
- ***Encapsulation:*** Ensuring that users of an object cannot change the internal state of the object in unexpected ways
- ***Polymorphism*:** According to which a name (such as a variable declaration) may denote objects of many different classes that are related by some common superclass
- ***Inheritance:*** Organizing and facilitates polymorphism and encapsulation by permitting objects to be defined and created that are specialized types of already-existing objects - these can share and extend their behavior without reimplementing that behavior

- Basic Object-Oriented Petri Nets (OOPNs) knowledge:

- ***Place, Transition and Arc:*** Can be viewed as objects are translated as a class definition.
- ***Token:*** Token in nets can not only be viewed as objects but also be represented as references to objects.
- ***Temporal Interval:*** Temporal interval is noted in each transition function.
- Input and Output places are ***encapsulated*** in the related Transition ***class***.
- ***Object:*** Each object is an ***instance*** of some class and consists of an instance of its object net and currently running instances of its method nets.
- ***Class:*** OOPNs consists of Petri nets organized in classes. A class defines a set of objects. A class consists of fields (a class component), actions (input, output actions), functions (parameters expression) and transitions (a

transition is an instance of a class consisting only of actions). Classes define how ***inheritance*** properties are propagated and provide mechanisms for configuration, reuse and sharing.

Moreover, new methods, ***constructors*** and ***predicates*** are added as well as places and transitions in object nets. These new class elements can be of course connected to the ***inherited*** or redefined elements.

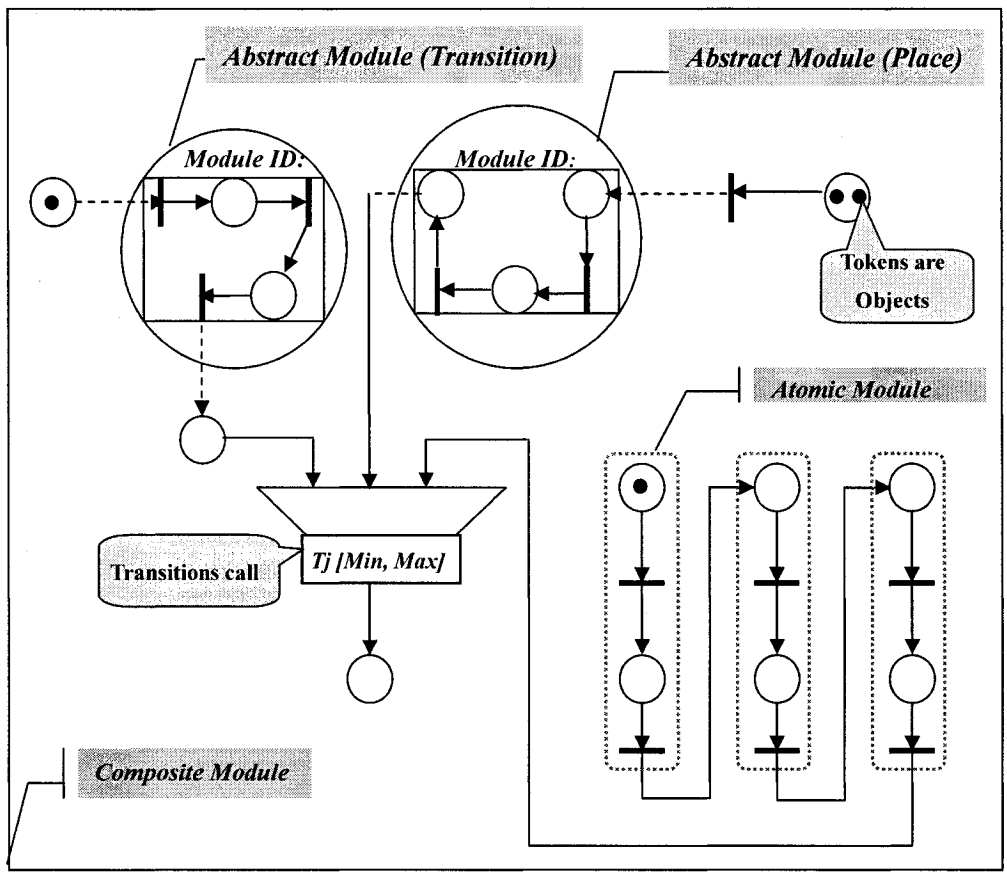


Figure 3.11 An Object-Orientation Petri Net Example

• ***Compositional techniques***

In the object oriented approach, every object is an instance of a particular class.

The inclusion of object-oriented concepts provides facilities for creating components that encapsulate behavior and state, and thereby supports:

- Abstraction;
- Refinement;
- Encapsulation;
- Reuse and sharing.

Components are described with classes that define how inheritance properties are propagated and provide mechanisms for configuration, reuse and sharing. A system is a composition of communicating components at every level of abstraction. During the design of a heterogeneous system classes are defined, refined, reused and configured. Refinement is supported by substituting less refined components with components containing an increased amount of detail. Abstraction and refinement allow complex systems to be modeled and successively refined

• ***Specification Language and Programming Tools***

Languages attempting to formalism OOP Ns have LOOPN++ ([LK94] [HB98]) and CO-OPN2 ([AC01] [HB98]) and so on. *LOOPN++* is a textual language for OOPNs demonstrates how object-orientation helps to address a number of typical protocol modeling issues. In the specification of LOOPN++, the class definition consists of three parts: "Fields" to define data, "Functions" to describe expression with parameters and operation, and "Actions" to represent the behavior of a system. The "Fields" part is a declaration of a token in Petri nets, and is used to represent the states of places. The "Functions" and "Actions" part together represent the transitions of Petri nets. Figure 3.12 shows an example of LOOPN++ for increment of a number. The "Export" phrase in Figure 3.12 is to define the externally accessible objects.

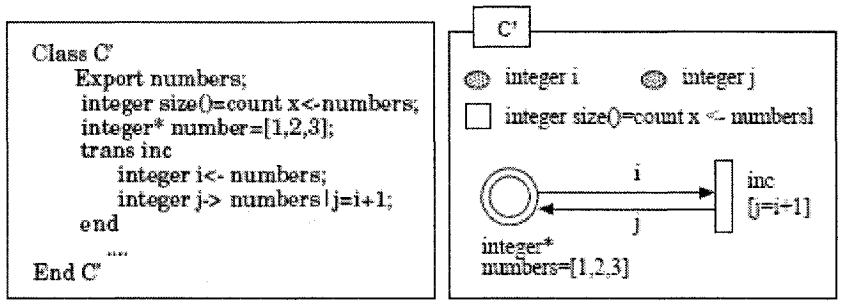


Figure 3.12 An Object-Orientation Petri Net Language Example

CO-OPN2 (Concurrent Object-Oriented Programming and Petri Nets) is a specification language based on algebraic Petri nets which permits an abstract description of concurrent operations and data structures. It is designed to specify and model the large systems. The class definition in CO-OPN2 consists of two parts: "Signature" part describe the interface with other classes, and "Body" part is to describe the internal behaviors operations of a class. CO-OPN2 supports composition by abstracting data type in order to reuse its type defined in other classes, and the method declared in "Signature" part is used as interface transition. For example, a CO-OPN2 for counting of clock (Figure 3.13)

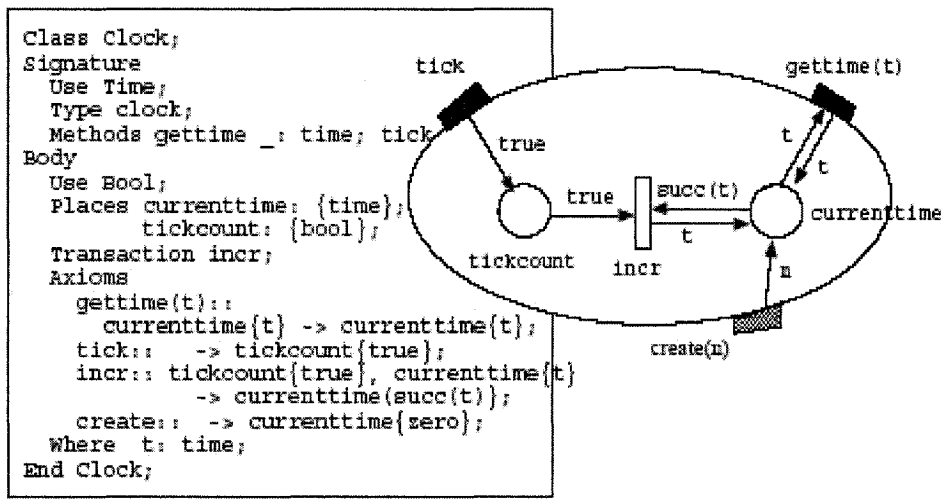


Figure 3.13 Concurrent Object-Oriented Programming and Petri Net Example

3.5.2.3 Advantages and Limitations

Using modular approach for compositionality allows the modeler to consider different parts of the model independently of one another. A modular approach to analysis is attractive: The modular state space is to alleviate the state space explosion problem, and then, allow prove PNs properties directly. As a result, modular approach often dramatically decreases the complexity of the analysis task, modular approach modeling makes larger systems easier to handle, and a large degree of reuse significantly decreases the development time and reduces maintenance cost.

• ***For Fusion Modular Approach:***

It is often the case that large PNs models contain a number of similar constructions, where only a few details differ. Modularisation alone does not allow for much reuse in such circumstances. However, by setting parameters for parts that differs between the modules we can construct a single module that describes the whole family of modules.

In the place fusion approach, it is relatively easy to incorporate ordering of messages over the same channel. But, the place fusion approach leads to an explosion of replicated places. When modeling a non-deterministic communication, the place fusion approach can handle fixed groups, but cannot handle an unspecified sender or receiver. On the contrary, the place unification approach can easily handle unspecified sender or receiver but requires extensions to handle groups.

• ***For Object-Orientation Modular Approach:***

"super places / transitions" is one of major characteristics of LOOPN++. It is used to represent the nesting structure of nets. It becomes a base to support the

abstraction of nets. LOOPN++ has a more regular syntax and is much more flexible. It has unified class hierarchy, allowing tokens and subnets to be arbitrarily intermixed. In this way, it provides ready support for models with multiple levels of activity. *LOOPN++* is a best object design environment, but it has the limitation of interactive editor. Furthermore, LOOPN++ does not fully reflect the actual concepts of objects because the nets include the global control structure of systems, and the token is only passive data type; Second, LOOPN++ tries to represent the abstraction by the feature of fusion only, but s not sufficient for abstraction of functional behavior and states; Third, LOOPN++ provides the "Export" phrase, but the message passing mechanism for the interaction among objects is not supported.

CO-OPN2 provides clear syntax and most of object orientation concepts which support inheritance and redefinition among classes. When each subnet of CO-OPN2 is integrated, interface methods (transitions) play a role links for the integration. However the unfolding mechanism of link parts, so the analysis and simulation method of CO-OPN2 is not clear, CO-OPN2 does not support the abstraction notion well. The limitation of CO-OPN2 is lack of firing interval description. Since Time Petri nets have widely used for the modeling and verification of time dependent systems and time parameter will affect the system performances and functional validity, OOPNs with time description becomes critical.

Finally, the problem is the size of model. If we hope to well apply OOPNs in the Object-oriented programming language (such as LOOPN++, CO-OPN2 and so on), the size of an object can become bigger than those we can manage.

3.5.3 Compositional Model using Algebra (M-nets, PBC)

High-level version algebra (M-nets) and low-level nets algebra (PBC) [FC01]

which using the concurrent programming language $B(PN)^2$ for compositional elementary Petri net semantics.

3.5.3.1 M-nets (Multilabelled nets) an algebra of high-level Petri nets

M-nets (Multilabelled nets), is an algebra of Petri nets with an applications to the semantics of concurrent programming languages [PE03]. A distinctive feature of this model is that it allows horizontal composition. It turns out that the composition operations of this domain have various algebraic properties. Moreover, the model is such that composition operations are coherent with unfolding, in the sense that the unfolding of a composite high-level net is the composition of the unfolding of its components. In the project of [BF95], it defines composition operations corresponding to the operators on M-nets:

- Parallel composition
- Choice
- Sequence
- Iteration
- Synchronization
- Restriction

Above correspondence between vertical and horizontal operations show how algebra can be used to define the semantics of a concurrent compositionally by programming language for Petri net semantics.

In addition, M-nets can be enriched by time intervals of duration attached to it, and are shown for advantage of hierarchical structure.

3.5.3.2 PBC (Petri Box Calculus) an algebra of low-level Petri nets

The PBC, called Petri net Box Calculus consists of an algebra expressions, and a corresponding algebra of boxes (a class of labelled Petri nets). PBC expressions are built from actions and operators and can be combined with other box expressions to form new expressions. In fact, it is always clear for an expression in a process algebra how it is constructed from sub-expressions.

In [AN93], Valmari proposes a compositional method which discusses the theoretical and technical prerequisites for compositional state space generation methods. This leads to the definition of a compositional semantic for nets (it provides a translation from box expressions to boxes). In [BD92], Best proposes the Box Calculus. The Box Calculus is intended to serve as a bridge between PNs theory and concurrent programming applications, offering an algebraic structure for PNs.

For a Petri net, PBC is usually used for how to be constructed from smaller parts. A standard static PBC expression is a word generated by the syntax:

Label: E::= a	*Variable:*X
*Sequence:*E;E	*Choice:*E[]E
*Concurrent Composition:*E||E	*Restriction:*E rs a
*Synchronisation:***E sy a**	*Relabelling of expression using function* f: E[f]

a is a constant, which is also known as a *basic action* or *multiaction.* PCB operators can be split into two groups. The three binary operators, sequence, choice, concurrent composition are *control flow operators.* The four unary operators, restriction, synchronization and relabelling, are *communication interface operators.*

In paper [KM03], Koutny extends the existing approach of the Petri Box Calculus (PBC) to a model, called the Time Petri Box Calculus (tPBC), which is

based on a class of time Petri nets, called ct-boxes, and their transition firing rule. tPBC can be used to specify and analyze concurrent systems with explicit timing information.

3.5.3.3 Advantages and Limitations

There are two advantages of M-nets, one is that carry additional information in their place and transition inscriptions to support composition operations. Another feature of M-nets, with respect to other classes of high-level Petri nets, is their full compositionality, thanks to their algebraic structure. As a consequence, an M-net can built out of sub-nets.

Because high-level and low-level algebra can be automatically generate by input B(PN)^2 programming language in the toolkit PEP [FC01], this compositional approach has advantage of simulating PNs' behavior or to model check it against some properties

However, one disadvantage of using algebra for helping start up composing PNs, we had to borrow the PEP-Tool which supports the elementary net semantics of B(PN)2, is no timers are implemented. Thus we had to implement the timer ourselves. A timer, however, consists of many multiple polling, leading to the problem that the simulation was very slow because the low-level Petri net generated by the tool for simulation reasons was very huge. In some cases the constructing composition PNs for time systems, it does not work "correct".

Another fact of limitation, such algebra techniques have been implemented in a PEP tool, unfortunately, it usually yields very large nets, as a necessary consequence of the fact that elementary nets were used.

3.5.4 Petri nets Composition with Trace theory

• ***Introduction***

Trace theory [RS83] is an algebra which was initially developed to manage about the design of digital circuits [KA86]. Apart from being able to specify a mechanism's operation, trace theory enables mechanisms to be composed into a system. Because the reachability tree of a Petri net is a state machine, it is possible to compose Petri nets using trace theory [CO95] and to subsequently find synchronization problems.

• ***Conceptions, Algorithm and Analysis Semantics***

Trace and Alphabets: A trace [CO95] is a sequence of symbols taken from an alphabet. The empty trace is represented as ε . For example, if $A = \{a,b\}$ is an alphabet then $t = ababb$ is a trace drawn from A. like follows:

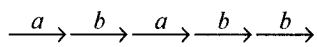


Figure 3.14 The trace ababb represented pictorially

Petri Net Composition using Trace theory: As a Petri Net executes, three related traces produced: a trace of markings, a trace of transition firings, and a trace of symbols. The operation of a Petri net can also be expressed as reachability tree, it shows all traces of markings and transition firings a Petri Net can produce. Reachability graph G is analogous to a reachability tree. For several interacting Petri nets, their reachability graphs can be composed using trace theory to give a composite reachability graph.

Joint Petri net Transition Firing Rules: Let $P = (P_p, T_p, F_p, \mu_p, \Sigma_p, \delta_p)$ and

$Q = (P_q, T_q, F_q, \mu_q, \Sigma q, \delta_q)$ be Petri nets. Also, let *tp* and *tq* be transition enabled by the current markings of *P* and *Q*. Then, *P* and *Q* will proceed in their joint executed according to the following rules:

- $\delta_p(t_p), \delta_q(t_q) \in \Sigma p \cap \Sigma q and \delta_p(t_p) = \delta_q(t_q) \Rightarrow t_p and t_q$ (Synchronization)
- $\delta_p(t_q) \in \Sigma q - \Sigma p \Rightarrow t_q$ (Parallelism)
- $\delta_p(t_p) \in \Sigma p - \Sigma q \Rightarrow t_p$ (Parallelism)

At any stage in the joint execution of two Petri nets, above rule must apply to avoid a deadlock condition. *The Composition algorithm* ➔ *Reachability Graph* ➔ *Petri nets*

Marking a Composite Reachability Graph: Reachability graph *G*, consists of a set of marking vectors, an alphabet of symbols, a*G*, and a marking to marking mapping function based on a symbol. If Petri nets are to be composed with trace theory then this will have to be accomplished through reachability graphs because explicit trace information, which is necessary for composition, is contained in tem. If *R* and *S* are reachability graphs, their composite can be created by performing the following algorithm. Upon completing the algorithm (Figure 3.15), a reachability graph specifying the joint behavior of *R* and *S* will have been made. The initial marking vector of the composite graph is the concatenation of the initial marking vectors of *R* and *S*.

```
For each successive marking change u→α v in R do
    For each successive marking change w →β x in S do
        If α,β ∈ aR ∩ aS and α=β then (u⊕w)→ α (v⊕x) endif
        If α,β ∈ aR - aS then (u⊕w)→ α (v⊕x) endif
        If α,β ∈ aS - aR then (u⊕w)→ β (v⊕x) endif
    Next
Next
```

Figure 3.15 Composite Reachability Graph Algorithm

Figure 3.16 and Figure 3.17 show an example of how the reachability graphs of two Petri nets are composed. From the composite reachability graph a Petri net can be constructed (Figure 3.17). Notice that where arcs are merged in the composite reachability graph, transitions are merged in the Petri net. That is, where two Petri nets must synchronize on a symbol, the transitions referring to the symbol, say *tp* and *tq*, are merged into a single transition, *tpq*. The pre-place set of *tpq* is the union of the pre-place sets of *tp* and *tq*, and correspondingly for the post-place set of *tpq*.

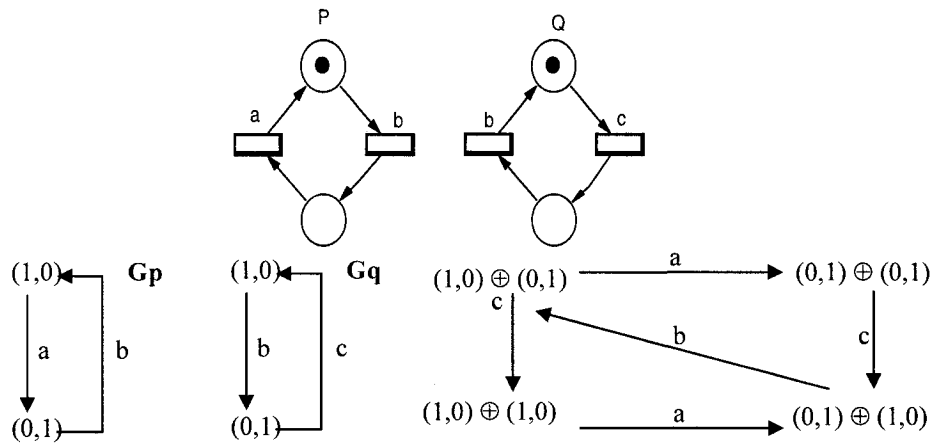


Figure 3.16 Petri Net's P and Q, their reachability graphs, and composite

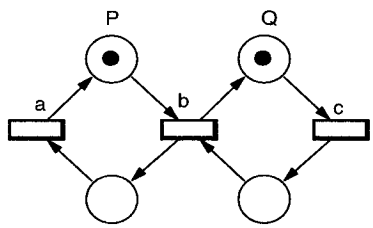


Figure 3.17 Petri Net's derived from the Reachability Graph of Figure 3.16

• ***Advantages and Limitations***

Trace Theory is used to describe the behavior of components in terms of

sequences (or traces). These traces can be composed to express cooperative behavior. It attacks the state explosion problem by avoiding the generation of a flat state space for the whole design.

Trace theory has limitation, if Petri nets are to be composed with trace theory then this will have to be accomplished through reachability graphs because explicit trace information, which is necessary for composition, is contained in them. The trace pertaining to a Petri net become apparently only when it is executed, producing a reachability tree.

3.6 Summary and Remark

Two main objectives of compositionality of a PN system are (1) Smaller the system size and (2) Re-use the information about the sub-components of a component to obtain information about the component's behavior.

Compositional Petri nets can be classified in different ways: Horizontal composition, Hierarchy and Modularization compositionality. They are achieved by one or more techniques such as elements fusion, folding, object-orientation, abstraction, refinement, algebra, trace theory and so on.

After review literatures and get the comparison of compositional Petri nets, hierarchical approach is selected in this project for structuring Time Petri nets for complex real time systems. Deeper description and a theoretical example are given in the next section.

3.7 Hierarchical Composition Time Petri nets

Hierarchical composition mechanism if the technique selected for Time Petri nets in this thesis. Following sections will give deep information from general introduction, conceptions, definition to related software tools and example.

3.7.1 Background Introduction

Because realistic systems tend to be complex and complicated, the ordinary representation may become too large to handle as well as state-space explosion problem. Compositionality is crucial for practical application of state-space explosion Petri Nets. Techniques mentioned in the last section work temp to realize compositionality. Among these solutions, in this project, a mechanism for hierarchical composition is selected so that the model may be constructed in a structured manner and composed of simpler units easily comprehensible by the designer at each description level.

There have been several approaches to the introduction of hierarchy into Petri Nets [CE01]. Valette [VA79] defines the concept of *block*, which is a refinement net with one initial transition and one final transition. The method for stepwise refinement and abstraction of nets presented in [SM83] is an elegant formulation to cope with the state explosion of PNs by transforming transitions and/or places into subnets and vice versa.

Current approaches, though dealing with the concept of hierarchy through sound formalisms, are not completely appropriate for time extended systems since the classical PNs model lacks essential notions like timing. An important contribution of this thesis is the definition of hierarchy for a modeling formalism

suitable for the design and verification of Time Petri Nets. In our approach timing is explicitly handled in the hierarchy. Furthermore, a convenient graph editor approach with good user interface which developed in this thesis will help designer construct TPNs by a composing a number of fully understandable entities instead of flat representation.

In this section, we firstly give conceptions and characters of hierarchy, then, a Hierarchical Compositional Time Petri Nets (HCTPNs) structuring example will be introduced. Finally, we will review the current designing tools and present our Graphical Editor and Verification Tool.

3.7.2 Conceptions and Basic Characters

Hierarchical Compositional structuring mechanisms have been introduced to handle system model complexity, associated difficulties of having too many details, and let a summarized view of the global system behavior be focused. Hierarchical concept supports two important mechanisms which let the components can be reused and shared. These two mechanisms are:

Abstraction/Refinement (top-down): The entities (Places, Transitions) should be able to represent abstract information at high-level decomposition structure. Using abstraction concept, we can focus on the essential and inherent aspects of an entity and ignore its accidental properties; *Coarsening (bottom-up)*: At a lower-level abstraction, should be refined.

Now, the Hierarchical Composition TPNs (HCTPNs) structure and modeling issues are addressed, while abstraction/refinement and coarsening mechanisms are interpreted in this chapter. HCTPNs differ from the ordinary TPNs in that they

include (see Figure 3.18).

Name	Description
Atomic place	Place that does not comprise any other TPNs
Atomic transition	Transition that does not comprise any other TPNs
Abstract place:	Can be named as composite place that specifies an underlying TPNs.
Abstract transition:	Can be named as composite transition that specifies an underlying TPNs.
Subnet:	A TPNs model which is degenerated into an action place or transition.
Supernet:	A TPNs model whose one or more elements (places, transitions) are abstract.
Atomic layer:	This layer distinguishes from composite layer mainly because it is concerned by net without any abstract elements
Composition layer:	At this layer, atomic and abstract places/transitions allow the hierarchical structure of composite components to be modeled.
Connection Points	Places or transitions used to connect different layer for passing events (or tokens). It consists of points for sending and points for receiving tokens.

Figure 3.18 Concepts of Hierarchical Composition Mechanism

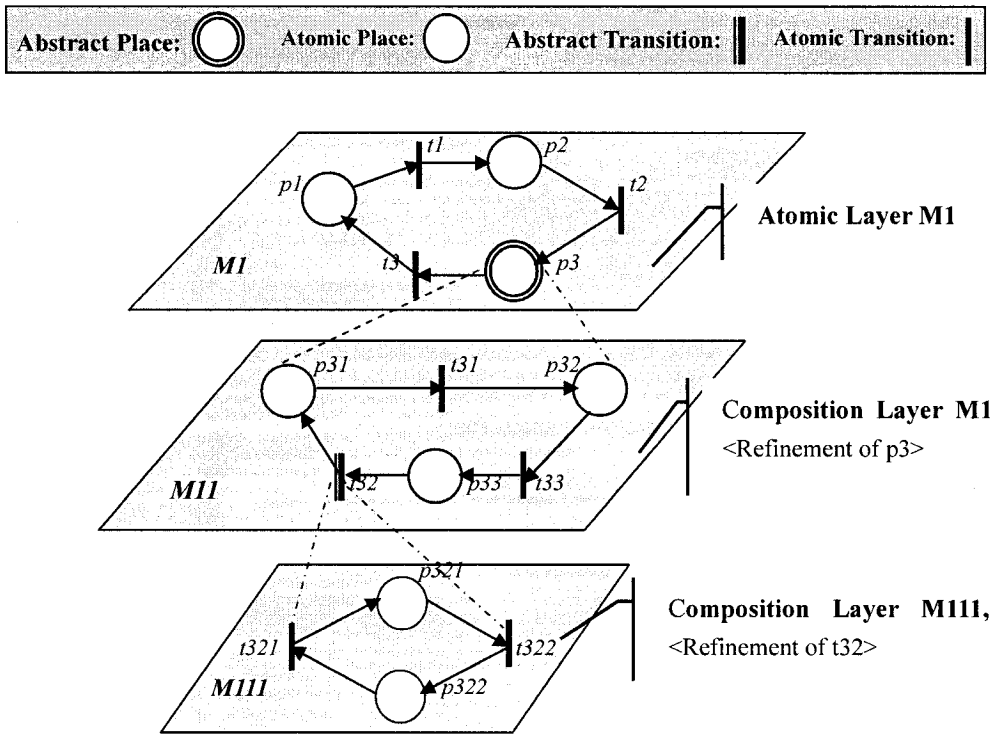


Figure 3.19 Illustration of hierarchical compositional Time Petri Nets

(The top hierarchy level is composed of M1, The lower level of abstraction is composed of M11, and M111 is the lower level of M11)

Composition rule in HCTPNs is represented by higher level abstractions each layer level net in the architecture. When passing events (or tokens) between the levels, lower level nets would have knowledge about the elements above them, but their details would be hidden from the higher levels.

3.7.3 Case Study

When we construct Petri nets by hierarchical composition of smaller components is a promising way to cope with the complexity of the design process for models of real time systems. A future compositionality must be on equivalences that are substitutive with respect to the operators. Practically important, such

equivalences allow compositional reduction techniques, where components may be replaced by smaller, but equivalent nets without affecting significant properties of the whole model. The equivalences are indeed substitutive with respect to more composition operators, the potential to exploit hierarchies in the model definition to obtain performance indices of truly large composite Petri nets by stepwise compositional reduction. We illustrate the effect of composition as well as compositional reduction by means of a 5 Dinging Philosophers example. This case study highlights the potential of compositional reduction.

3.7.3.1 Introduction

Five Dining philosophers is a common computing problem, used to demonstrate deadlock. This Petri net models 5 philosophers who are dining together. The philosophers each have a chopstick next to them, both of which they need in order to eat. As there are only five chopsticks it is not possible for all 5 philosophers to be eating at the same time. Flat level Petri net of this problem is displayed in Figure 3.20. In flat level design, each of the philosophers has the same structure enabling them to be represented by a single Petri net (Figure 3.21). This Petri net can then be added to the system as a subnet for each of the philosophers (Figure 3.22).

3.7.3.2 Design and Specification

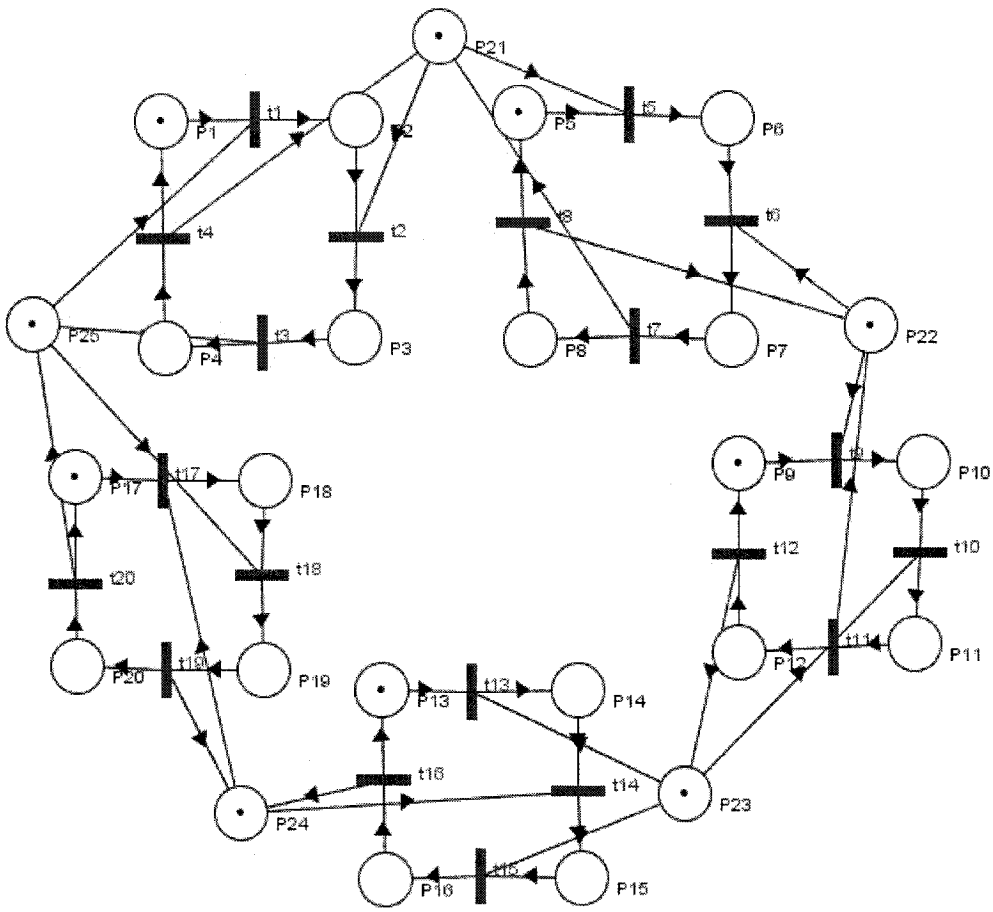


Figure 3.20 Single level of Dining Philosophers Structure

Figure 3.20 shows the traditional Dining Philosophers design as flat level. Places P21, P22, P23, P24 and P25 represent the chopsticks. Every rectangle area represents one philosopher, each has four places and four transitions. For example, the status of every philosopher begin from thinking (p1)→picking up his left hand chopstick (t1)→ the state when this chopstick has been picked up(p2) →picking up his right hand chopstick(t2)→eating(p3)→releasing his left hand chopstick(t3)→ the state when this chopstick has been released(p4) →releasing his right hand chopstick(t4)→thinking(p1).

Dining philosopher is a simple case that you can describe it in a flat level, however, you will find it is a tedious repetition designing or editing work. The prominent advantage of hierarchical compositionality is reusing modules, and simplifying the design (Figure 3.21).

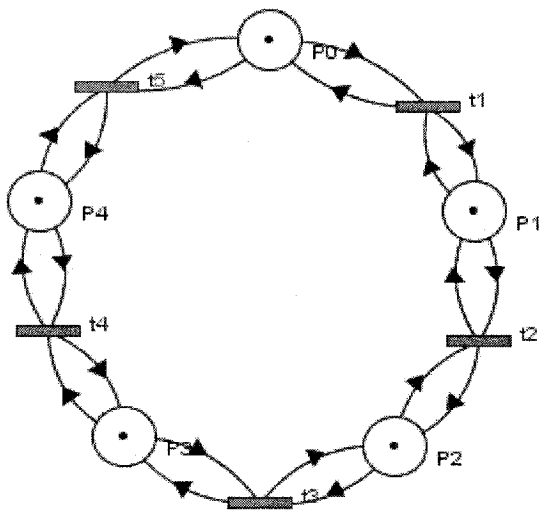


Figure 3.21 Hierarchical Dining Philosophers Structure

Figure 3.22 presents a hierarchical design of the dining philosopher problem. Each philosopher is represented by a subnet, an atomic transaction change to be abstract transaction when it is denoted subnet ID (t5_0, t5_1, t5_2, t5_3).

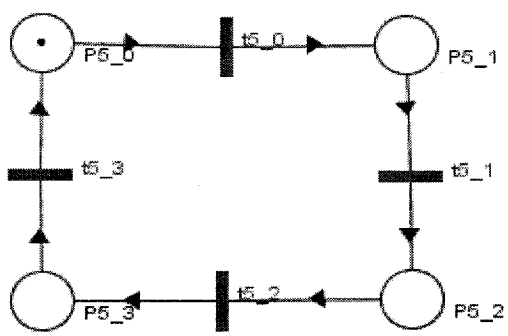


Figure 3.22 Subnet of Hierarchical Dining Philosophers Structure

3.7.4 Petri NetsTools for Hierarchy Development Background and Objective

There are approximately fifty existing tools for Petri Nets in Database (http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html). The editors studied include ARP, INA, TINA, Visual Object Net ++, PNtalk, PEP-Tool and so on. The functionality provided varies capabilities such as Graphical Editor, Token Game Animation, Performance Analysis, Place/ Transition Invariants, Workflow Management, State Spaces. A sample of the editors available have been obtained and investigated to identify the capabilities common to Petri net Editors. These techniques help to design Petri nets by providing some useful tools for place-transition, timed, and extended timed Petri nets.

The features offered by existing Petri nets editors which for Time Petri nets or hierarchical editor are also investigated.

- *ARP* (Portuguese for "Petri Net Analyzer and Simulator") tool is a program for helping design with Petri nets. It provides some useful tools for place-transition, timed, and extended timed Petri nets, in which firing intervals $[t_{min}, t_{max}]$ are associated to each transition (Merlin's approach). It's structure is very modular and provides a simple and intuitive user interface, using textual windows. The program structure is modular and allows to easily design and integrate new tools for Petri net analysis or manipulation.

- *TINA* is an Enumerative Approach for analyzing Time Petri Nets to propose for TPN a technique for modeling the behavior and analyzing the properties of timed systems. The motivation is specifying and proving correctness of time-dependent systems.

- *INA* is a tool package supporting the analysis of Place/Transition Nets (Petri Nets) and Colored Petri nets. INA combines the following: a textual editor for nets, a by-hand simulation part, a reduction part for Place/Transition-nets, an analysis part to compute, a model checker for the Computational Tree Logic (CTL). The editor gives the possibility to construct and to edit nets. One can use any graphical editor, too, if one knows the structure of its database file, by writing a program which converts to the INA-file format. The by-hand simulation part allows starting at a given marking to forward fire either single transitions or maximal steps; the user can thus traverse parts of the reachability graph. The reduction part can be used to reduce the size of a net (and of its reachability graph) whilst preserving liveness and boundedness.

- *PED* is a hierarchical Petri net editor provides comfortable and easy-to-use functionalities for the construction of hierarchical place/transition nets. An assignment of additional attributes like priorities, capacities, time durations and time intervals to places, transitions, and arcs is possible. By this way, many related Petri net classes, especially those which are analyzable with INA, are supported. This includes net classed with non-stochastic time assignments like time nets, timed nets, and duration interval nets.

- *THORN/DE* is a general-purpose, graphical, discrete-event simulation tool based on a special class of high-level Petri Nets called Timed Hierarchical Object-Related Nets. THORNs allow the specification of individual tokens, they provide delay times and firing durations for transitions, and THORN models can be hierarchically structured with respect to transition refinement and subnet invocation. Token types and transition inscriptions are specified by C++ code.

However, these Petri net tools provide only limited support for characteristics combine ***Time extend analysis, Graphical edit and Hierarchical design***. Some tools can analyze time interval but based on textual editor, making them relatively difficult to define, whereas Petri nets are easily expressed in a simple graphical way. If these tools have graphical edit and time interval analysis functions, they don't have hierarchical structure construction property, where by a system is decomposed into a set of Petri nets rather than a single net. Such approaches simplify system design, because different components of a system can be designed and modeled individually. We tried to find a tool that provides a simple and intuitive user interface, using graphical windows to edit Petri nets with hierarchical concept supports both top-down (refinement) and bottom-up (coarsening) net design. An integrated hierarchy browser provides many editing and navigation features.

The following chapter will describe the tool that we select and program to implement hierarchical composition Petri nets.

CHAPTER 4

TOOL DEVELOPMENT AND IMPLEMENTATION

The previous chapters have stated the aims of this project and the current features present in Petri net tools. To achieve both hierarchical composition design editors and verification, we introduce *GraphLab* tool to realize structure construction and model checking for or Time Petri nets in this project. *GraphLab is* a toolbox for analysis of (time) Petri nets, which constructs state class graphs (abstract models) and exploits them for LTL, CTL. The original version of this tool is developed in the Research Lab of Vérification des systèmes temps reel of École Polytechnique de Montréal, This project modify and add new functions into the *GraphLab* tool.

This chapter is organized as follows: first briefly introduce the general information of graphical user interface of *GraphLab.* Then, explain what are improved for hierarchical structure editing in detail. Improvements include interfaces of *GraphLab* and its software architecture: user interface, editing facilities, save and input/output file formats. Last in this chapter recalls concisely the classical state classes technique, that preserves linear time properties of the concrete state space, and preserving clock and bound domains time properties, also include parameters options during construction state class graph.

4.1 Design Based on GraphLab

Many existing tools are operating system specific – in fact most are limited to

specific brands of UNIX (e.g. Linux, Solaris), *GraphLab* is a tool portable to operate on a wide range of Windows and UNIX platforms. *GraphLab* offers a graphics interface similar to a graph editor with some specific functionalities related to the real time field. This section introduces *GraphLab* integrated development environment (IDE). It also describes the design tools and the way to use them.

4.1.1 Basic Features

The tool should provide the basic features necessary to design and edit a Petri net. Support for hierarchical Petri nets (4.2) and State Class Graph generation (4.3) are of greater importance to the project. Places, transitions and arcs should be able to have their properties such as token number, firing time interval and weights modified. It should be possible to change the position of components on the net and to have the ability to delete components.

Graph Browsers (Figure 4.1) is the user interface of *GraphLab* whit a window where most of the development functions on a graph are performed: editing, visual designing, navigating and browsing. A Graph browser contains several panes for performing these development functions. The following picture (Figure 4.1) shows *GraphLab* and gives a denomination of most of the tools used to perform the design tasks:

- *Graphical User interface:* The *GraphLab* toolbox is built in a modular way. These modules can be used independently or in combination. Modules include:
  - A graphic editor for Petri nets, Time Petri nets drawing facilities;
  - A tool for building state space abstractions, calculating the number of vertices and links, time-consuming implementing all the constructions of the state class graph.

- *File format: GraphLab* editor produces Petri net file (XML format) that can be later read by the state space construction in the state class graph view function. This editor is able to edit and draw their outputs. The graphical input format is simple and intuitive. Several output formats are available, including XML *GraphLab* files (.xml format) and the *Tina* tool files (.tina format) for use.

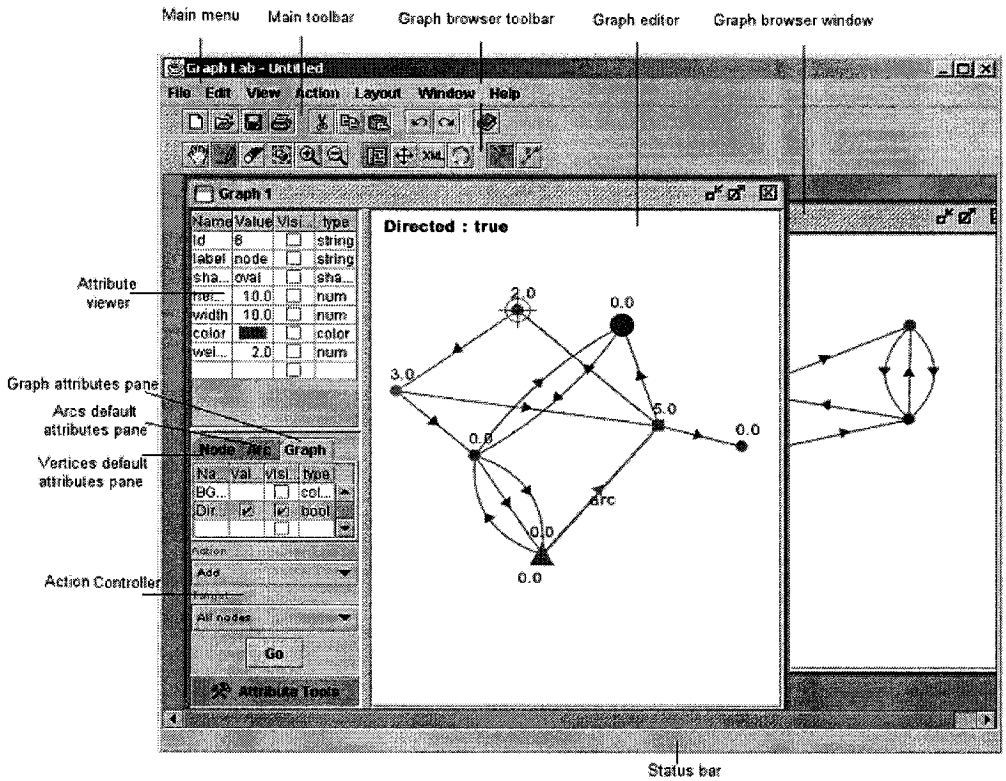


Figure 4.1 GraphLab User Interface Overview

There are two tool bars (Figure 4.2): the Main toolbar and Graph browser toolbar. The Main toolbar is displayed at the top of the GraphLab MDI window under the menu bar. It is composed of small buttons grouped by functionality: File, Edit and Help. You can modify the toolbars display by checking or unchecking selections on the View menu. The Graph toolbar buttons deal with Graph design and representation actions. The toolbars can be moved from one site of GraphLab MDI

window to another side by dragging them from the handles at their left side and then positioning then on the desired side. They can also be made floating by dropping them on any place, but the sides, to gain more working space.

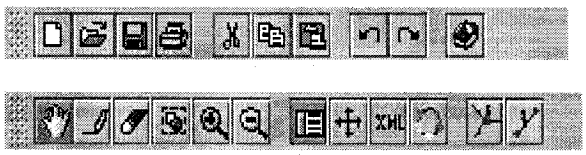


Figure 4.2 Main toolbar and Graph browser toolbar of GraphLab

The attribute viewer edits the list of attributes of the graph component (arc or node) in focus. The graph component in focus is distinguished from the other graph components by a special cursor called the focus cursor, as shown on the following picture (figure 4.3). By default, the attribute viewer is not visible. To make it visible you need to toggle the Attribute viewer button in the graph browser toolbar or in the view Menu. The attribute viewer displays the list of attributes of the graph component in focus in a table with four columns (Name, Value, Visible, Type) each row represents a separate attribute. There are 5 Types of attributes: String, Number, Shape, Color and Path. The usage of some types is restricted to some graph components types (arcs, nodes, graph).

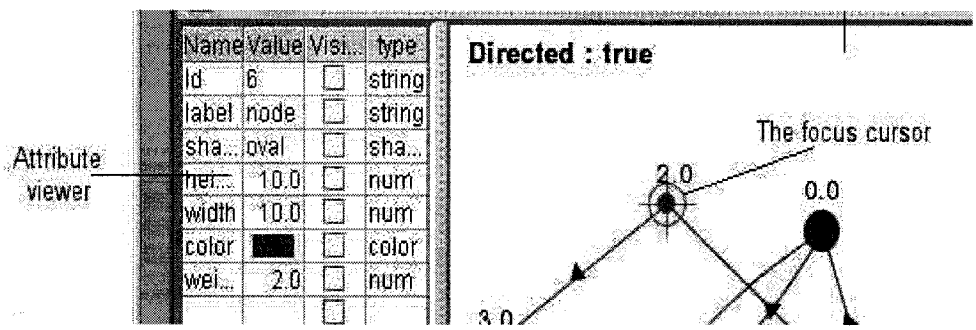


Figure 4.3 Attribute viewer of *GraphLab*

The Graph editor is the place where you design your graph using the tools offered in the Graph browser toolbar or the action menu. The design process of a graph passes by the creation of nodes and arcs. When the move button is pressed, you can move graph components (nodes or arcs) by simply moving the mouse cursor on the component you want to move, press the left mouse button and drag the component where you want to, and finally release the mouse button. You can keep doing this action as long as the move bottom is pressed. When you move the mouse cursor on a graph component the mouse cursor becomes a cross. For an arc you need to move the mouse cursor on its arrow if the graph is directed, or on the circle that replaces it if the arc is not directed

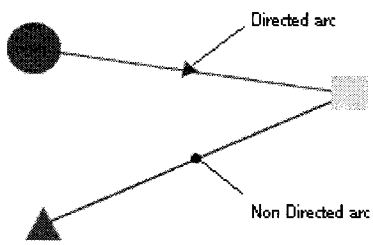


Figure 4.4 Move Action of *GraphLab*

When the Draw button is pressed, you can create nodes and connect them with arcs. To create a new node, you need to go to an empty area in the graph editor and press the left mouse button. A new node will be created with a set of attributes equal to the one listed in the Nodes defaults attribute pane. To create an arc you need first, to move the mouse cursor on an existing node. When the mouse cursor becomes a cross, press the left mouse button and drag the mouse on to the second node you want to connect to the first one. Finally, release the mouse button. The new arc will be created with a set of attributes equal to the one listed in the Arcs default attributes pane. Note: if you drag and release the mouse bottom on an empty area, a new node will be created and the newly created arc will connect the first node to this new one.

4.1.2 Model Checking Realization

Graphlab allows the construction of the so-called state class graph - that is a graph where timed states are symbolically represented by a marking and a firing domain. For Time Petri nets, various linear Model checking linear temporal logic (LTL) properties for state class graph constructions, these properties include LTL (Bound domains and Clock domains) and Clock CTL (clock domains). In order to get different result of state class graph for analysis, like figure 4.5, user can indicate the number of states to compute and visible of transitions and places.

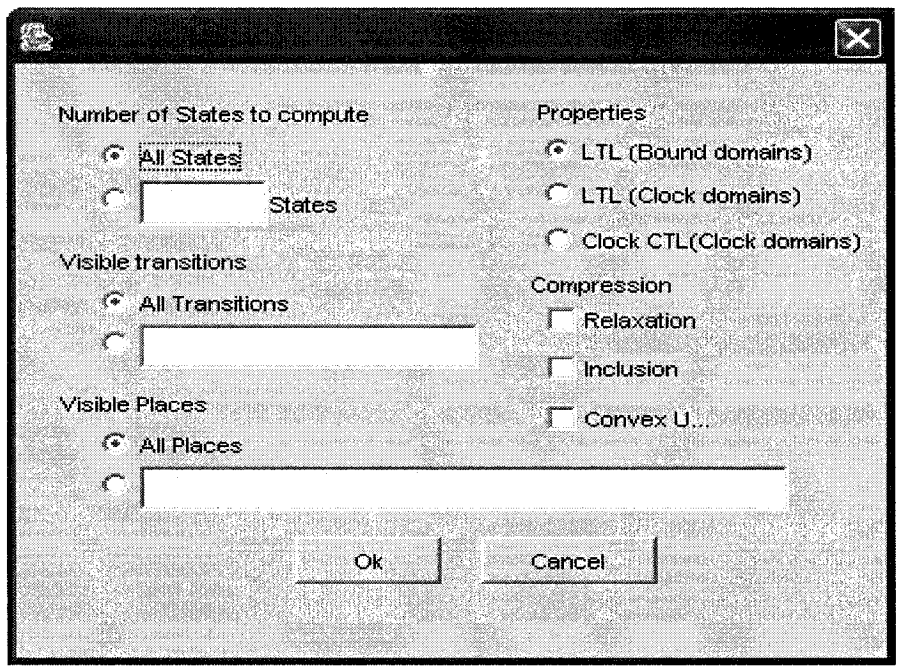


Figure 4.5 Linear Temporal Logic (LTL) properties for state class graph

4.2 JAVA Programming Modification for Hierarchical Capabilities Editing

The aim in this project is hope to find a tool for hierarchical composition Time Petri nets structure construction, moreover this construction must be satisfy the verification, in other words, all kinds of original properties can not be changed. In

order to achieve this purpose, *GraphLab* Tool is modified and added components and functions for hierarchy. The whole programming work use Java Object Orientation Language.

4.2.1 Graphical User Interface Modification for Hierarchical Composition

GraphLab graphical user interface requires all the features that any typical graphical user interface requires, such as menus, toolbars and drawing area which provide features to edit Petri nets, such as moving the position of components and modifying components' properties such as token number, arc weights and time interval. The section describes the elements and new components add in the user interface .The design or drawing area will form the largest area of the user interface, it is here that Petri nets will be created and edited. To perform functions in the design area one of the options such as transition or edit will be selected. These options will be displayed in a toolbar, preferably adjacent to the drawing area.

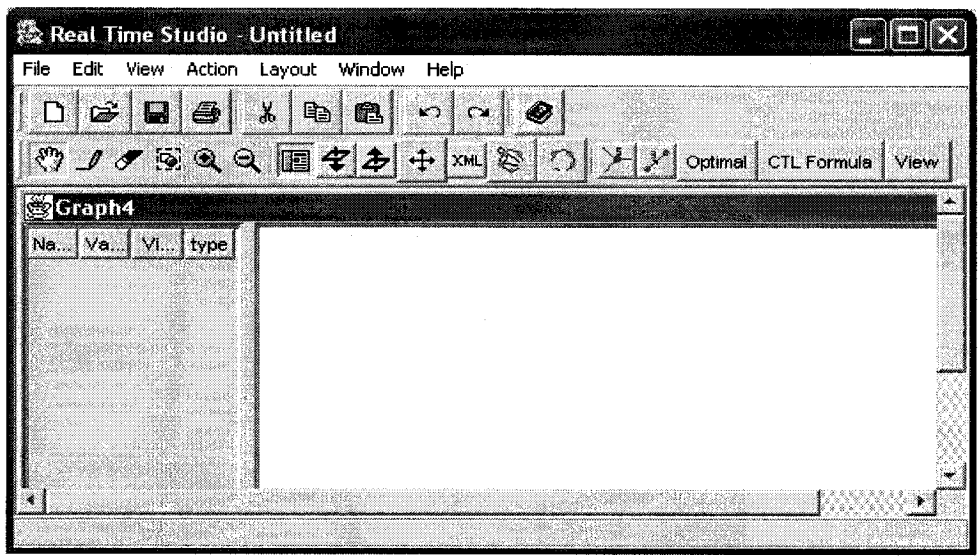


Figure 4.6 *GraphLab*-Graphical Editor for Hierarchical Time Petri nets

- *Tool Bars*

Main toolbar provides file opening and saving options.

Graph browser toolbar add two new buttons:

: Edit or show the subnet hierarchical structure.

: Edit or show the supernet hierarchical structure.

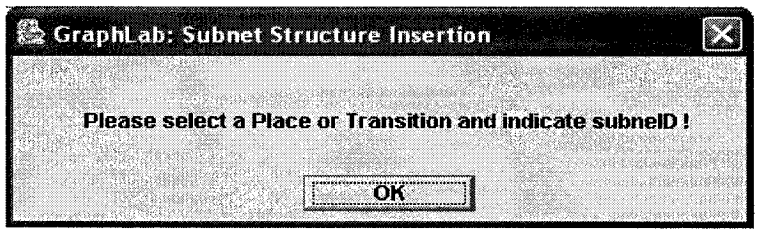


Figure 4.7 Dialog message when creating abstract element without subnetID

When an atomic place or transition will be expanded to it's subnet, user click button. If this place or transition isn't named a subnet ID in *Attributes viewer* previously, Dialog message (figure 4.7) will give hint to user to indicate it's subnet ID description. To edit the contents of a subnet, click on the subnet, this will display the contents of the Petri net for editing. If this level contains subnets they can also be clicked on to move further down the hierarchy.

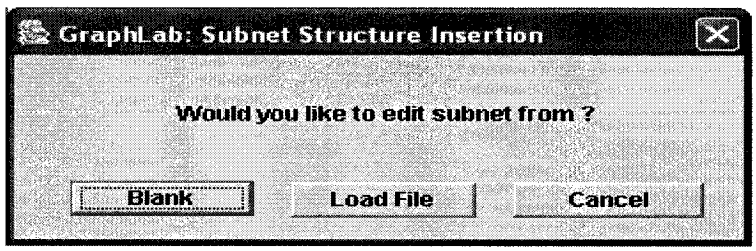


Figure 4.8 Dialog message when creating an abstract element with subnetID

When an atomic place or transition will be expanded to it's subnet, user click button. If this place or transition is named a subnet ID in *Attributes viewer* previously, Dialog message (figure 4.8) will give a hint to user to the path of open

the file: New a blank file to draw Petri nets (subnet); Load an existing file from a path; Last one – cancellation. When back button is clicked, it is used to move back up the hierarchy.

- *Status Bar*

A status bar will be placed at the bottom of the user interface to provide information to the user.

- *Attributes viewer*

Some editing features such as changing arc weights or transition firing interval cannot be carried out in the drawing area, because input needs to be obtained from the user. *Attribute viewer* (figure 4.9) is used to obtain such information. By default, the *Attribute viewer* is not visible. To make it visible you need to toggle the *Attribute viewer* button in the graph browser toolbar or in the view Menu. The *Attribute viewer* displays the list of attributes of the graph component in focus in a table with four columns (Name, Value, Visible, Type) . Each row represents an a separate attribute,

- The name of the attribute is displayed in the column "Name": In this column, new row elements (subnetID, supernetID, connectTo, connectFrom) are added for hierarchical structure design.

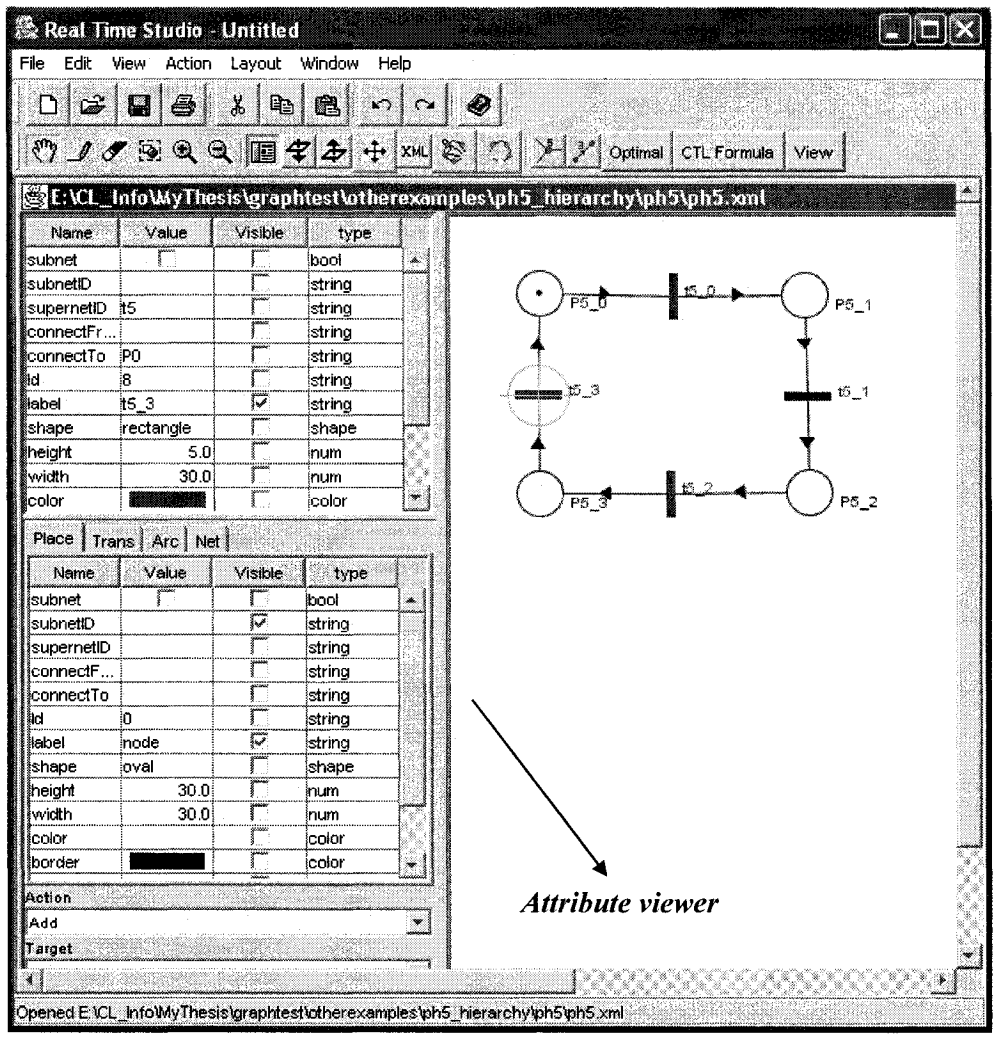


Figure 4.9 Attributer viewer of *GraphLab*

- *subnetID*: If the description is indicated to an atomic place/transition. The change will turn to green color automatically in drawing area. Once this attribute is filled, the same name directory is created automatically when user save file (Details explanation given in 4.2.3). Blank means there is no subnet of the place/transition, otherwise, it is an abstract place/transition. For example, 5 dining philosophers of 3.3.2 section.

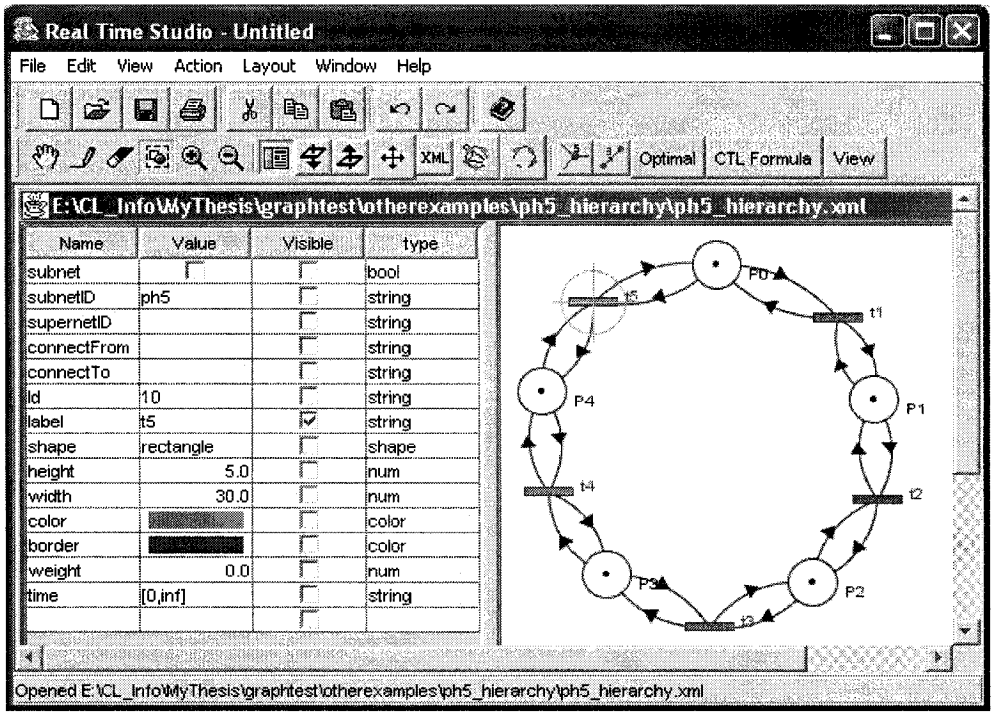


Figure 4.10 Hierarchical Dining Philosophers Structure and Attribute Viewer

Like figure 4.10, presents a hierarchical design of the dining philosopher problem. Each philosopher is represented by a subnet, an atomic place/transaction change to be abstract place/transaction when it is denoted subnet ID in the attribute (ph5 in above example), the color change to green automatically to distinguish them from the other components of a Petri net. t1, t2, t3, t4, t5 are abstract transitions, they generalize dining philosophers' thinking, eating statuses.

- *supernetID*: If the description is indicated in this row at the "Name " column, the Petri nets in the drawing area must be a subnet of an abstract element which supernetID named (For example, in figure 4.9, details of each philosopher are expanded in drawing area, it's supernetID is t5).

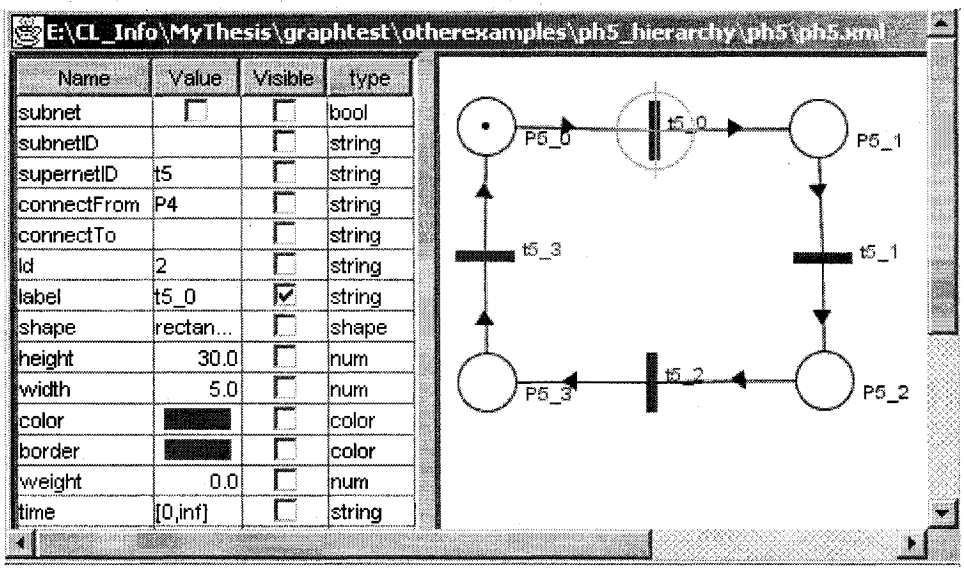


Figure 4.11 One Subnet of Hierarchical Dining Philosophers Structure

- *connectTo and connectFrom:* A subnets interact with the outside of the subnet using *connectTo* and *connect From*. This enables arcs from the current level of the Petri net to be connected to the interaction points and thus interact with the subnet. For example, figure 4.11 shows the inside graph of a subnet. t5_0, t5_1, t5_2, t5_3 represent picking up and releasing left and right chopstick. They are ***Connection Points*** between subnet and its high level nets.

Drawing area	***Connection Points***	***means***
Place	connectTo: t1,t2,...tn	place(subnet) → transitions1,2,...,n (abstract of high level)
	connectFrom: t1,t2,...tn	transitions1,2,...,n (abstract of high level) → place(subnet)
Transition	connectTo: p1,p2,...pn	transitions1,2,...,n (subnet) → place(abstract of high level)
	connectFrom: p1,p2,...pn	place(abstract of high level) → transitions1,2,...,n (subnet)

Figure 4.12 Connection Implementation for Different Level

Use example in 3.3.2 section, one subnet of 5 dining philosophers. Like figure 4.12. supernetID is given as t5 and all the *connectFrom* or *connectTo* points

are use to transfer arcs connection information between lower-level and higher-level. t5_0 or t5_1 has connectFrom point P4 or P0 from supernet; t5_2 or t5_3 has connectTO point P4 or P0 from supernet

- t5_0 is connected to Chopstick5 of high level. The token in Chopstick5 via t5_0 to P0;
- t5_1 is connected to Chopstick1 of high level. The token in Chopstick1 via t5_1 to P1;
- t5_2 is connected to Chopstick5 of high level. The token in P1 via t5_2 to Chopstick5;
- t5_3 is connected to Chopstick1 of high level. The token in P2 via t5_2 to Chopstick1.

- *Value:* The value of the attribute is displayed in the column "Value". You can view the value of the attribute right under the corresponding graph component by checking the check box in the columns "visible".

- *Type:* The Type of the attribute is displayed in the column of *Attribute viewer.* There are 5 Types of attributes: String, Number, Shape, Color and Path. The usage of some types is restricted to some graph components types (arcs, nodes, graph). The following table gives a small description of each type and the graph component it may be used in.

- *Weight:* A specific value is assigned to each arc as a weight *w*. An arc with weight *w* enables the transition to fire only if the content of the place at the source of the arc is less than or equal to *w*.

- *Token:* Amounts of tokens (integer) can bedefined for place selected.

- *Time*: Time constraints are indicated in *Attribute viewer.* Time constraints like

delay between events, time intervals for handling messages, response time above threshold, latency. Time constraint value in *GraphLab* is [0, inf] (here, infinite is abbreviated to inf). While all time constrains equal [0, inf], there is no limit on the number of simultaneous firings of the transition. Similarly, if a transition is enabled `several times' $[n_1, n_2]$ (i.e., it remains enabled after initiating a firing), it may start several independent firings in the same time instant. If the firing times of some transitions may be equal to zero [0, 0], which means that the firings are instantaneous; all such transitions are called immediate.

Summarily, *GraphLab* tool has ability to support hierarchical Petri nets such that interactions between different levels are explicit, and can be switched between effectively. For example clicking on a subnet could cause it to be loaded and displayed by the editor. Connection points could be identified by painting them a different color (i.e., green for abstract places or transitions) and by displaying them on the *attribute viewer* list of the subnet, to show how the Petri net interacts with subnets. When subnets are introduced users should be able to select what Petri net the subnet contains. There should further be the choice to save the subnet using the already selected file or to save it under a different name specifically for use in the current design.

4.2.2 File Format Design Modification and Directory Creation

4.2.2.1 Using XML File Format

GraphLab uses the Extensible Graph Markup and Modeling language (XGMML) for its internal representation of graphs. XGMML is an XML 1.0 application specifically designed to deal with graphs. This representation has been preferred over other representations due to its power of expression, extensibility and its wide acceptance as a standard. *GraphLab* provides a simple XML format for

specifying places, transitions and arcs, tokens, time constraints.

However, none of the proposed Petri net interchangeable file formats (mention in section 3.4) and *GraphLab* support hierarchical Petri nets. It was therefore necessary to design a file format that closely follows the proposals but also provides a format to save hierarchical Petri nets.

A natural way to save a hierarchical Petri net is for each net to have its contents saved in a separate file. Where a Petri net includes a subnet, the subnet can be specified by providing the file name for the subnet. In order to save or load a hierarchical Petri net file correctly, it was therefore necessary to add new description for attributes such as arc connections, subnetID, supernetID in the file format of XML. Different levels of hierarchy would be saved as different XML files, making them more accessible to other tools. One of the proposed Petri net interchange file formats, a simple example is shown in figure 4.12. The format for the arc was not altered. The format for places and transitions needed modifying to identify if they are abstract elements with subnet and connection points with other level.

A simple example of figure 4.10 is shown in figure 4.13 by select button XML at tool bar.

```
<?xml version="1.0"?>
<graph directed="1">
<node id="1" label="P0" token="1" place="true" >
        <graphics type="oval" x="134" y="24" w="30" h="30" fill="#-1">
        </graphics>
</node>
<node id="2" label="t1" token="0" place="false" subnetID="ph1" time="[0,inf]" >
      <graphics type="rectangle" x="209" y="58" w="30" h="5" fill="#-16711936">
      </graphics>
</node>
    ......
<node id="4" label="t2" token="0" place="false" subnetID="ph2" time="[0,inf]" >
<graphics type="rectangle" x="233" y="175" w="30" h="5" fill="#-16711936">
</graphics>
</node>
    ......
<edge id="1" source="1" target="2" label="arc" token="1" >
    <graphics type="line" stroke="1" outline="#-16776961">
    </graphics>
</edge>
    ......
</graph>
```

Transition t1 is an abstract with low level net

position

Source =1, get p0;
Target=2, get t1

Figure 4.13 Example of a Petri Net Markup Language File.

(The above sample file specifies places, transitions, arcs, subnetID and time interval from the example of figure 4.10. The Petri Net markup language allows a label (name), position to be saved, while places also have their initial marking (token number) saved. The source and target of an arc, its start and finish place or transition and the arc weight are saved within the Arc tag.)

A simple example of figure 4.9 is shown in figure 4.14 by select button XML at tool bar.

```
<?xml version="1.0"?>
<graph directed="1">
   <node id="1" label="P5_0" token="1" place="true" supernetID="t5" >
      <graphics type="oval" x="58" y="51" w="30" h="30" fill="#-1">
      </graphics>
   </node>
    <node id="2" label="t5_0" token="0" place="false" supernetID="t5"
connectFrom="P4" time="[0,inf]" >
      <graphics type="rectangle" x="146" y="53" w="5" h="30" fill="#-65536">
      </graphics>
   </node>
    ... ...
```

Figure 4.14 Example of a Petri Net Markup Language File.

(The above sample file specifies places, transitions, arcs, supernetID and time interval from the example of figure 4.9. The Petri Net markup language allows a label (name), position to be saved, while places also have their initial marking (token number) saved.)

4.2.2.1 Directory Creation and XML File Saving

In order to realize hierarchical structure design in GUI of *GraphLab*, New programming is added.

First modification is automatically directory creation when a design is saved after a user finish drawing a Petri net at drawing area. This new directory has a same name as *subnetID* at *Attribute viewer*. If this Petri net has several different name of subnetID, it will get same amount and same name of new directory, they are sub-directories of the directory which storage the original file. We call this is parent and children relation, and this relationship make different levels in hierarchy clear. If places or transitions of the subnet are abstract elements which can be expanded a subnet, we can say it's grand-children net of the original file.

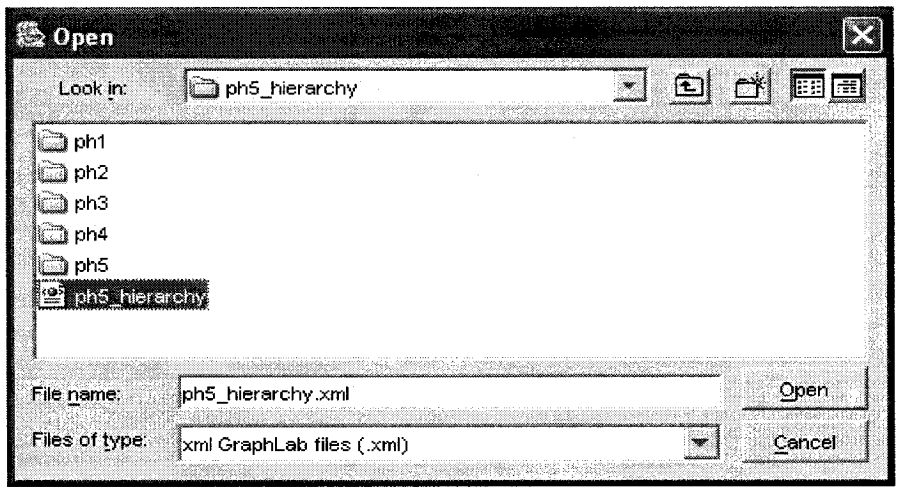


Figure 4.15 Sub-directories creation (hierarchical 5 dining philosophers)

When drawing the first level of 5 dining philosophers (figure 4.10), 5 subnetID are defined to abstract thinking, picking chopsticks, eating and releasing chopsticks statuses of 5 philosophers. Save action at this level will create ph1, ph2, ph3, ph4 and ph5 directories. Each directory has the same name of it's XML file. User can design subnet file in the relevant directory. Figure 4.16 shows ph5.XML file in the same name directory which belong to child level directories.

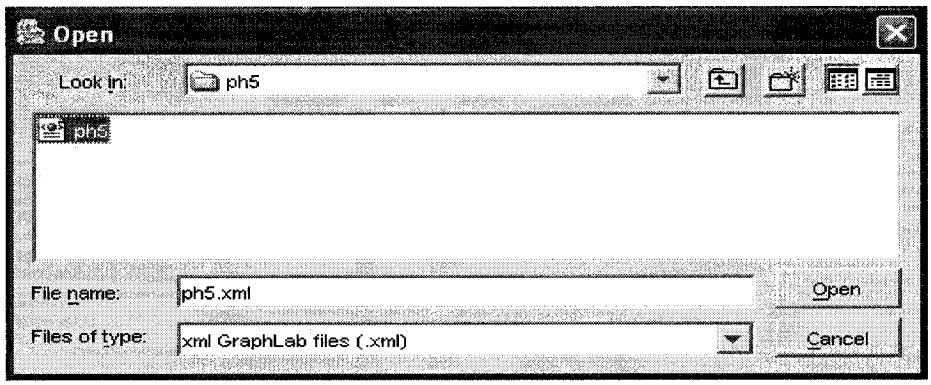


Figure 4.16 Sub-directory and it's XML file in 5 dining philosophers example

This file saving and directories creation mechanism provides technique for Petri nets structure specification both by top-down and bottom-up.

4.3 State Class Graph and Preserving LTL Properties

This section is organized as follows. Section 4.3.1 reviews the terminology and important conceptions for time Petri nets. Section 4.3.2 introduces the available approaches for building state space abstractions (state class graph) that preserve LTL and CTL properties in *GraphLab* and an example is gave.

4.3.1 Important conceptions using in *GraphLab*

Beside the common graphic editing facilities for hierarchical composition, the software tool *GraphLab* proposes abstractions operate either on untimed ones (represented by Petri nets) or timed systems (represented by Time Petri nets). For untimed systems, computation of abstract state spaces helps to prevent combinatorial explosion. For Time Petri nets, that have in general infinite state spaces, they provide finite symbolic representation of their behavior, in terms of state classes.

State classes: As transitions may fire at any time in their temporal intervals, the states of a Time Petri nets generally admit an infinity of successors by the timed reachability relation. Any finite representation of this state space must thus rely on some agglomeration of states. Those agglomerations are called *state classes* (details see section 2.1.4).

State space: The basic idea behind a state space is to construct a directed graph, which has a node for each reachable marking and arcs corresponding to occurring binding elements. State spaces are also called occurrence graphs or reachability graphs/trees. The first of these names reflects the fact that a state space contains all the possible occurrence sequences, while the two latter names reflect that the state space contains all reachable markings. From the state space it is possible to analyze and verify an abundance of properties of the system. For Time Petri nets, state class

graph is abstraction of state space construction.

State class: Detail information can be found in section 2.1.4.3. State class is a technique for reachability analysis of time Petri nets has been available for a long time [BM83, BD91]. This method computes state classes. Which are finite representations for the state graphs of bounded time Petri nets. State classes capture a marking and a convex firing time space for the transitions enabled at that marking. It is the cover of the state space equipped with a transition relation satisfying property. Note that transitions between state classes graph are not timed anymore, state classes and their reachability relation allows one to abstract time from the behavior of a net. *GraphLab* offers abstract state space constructions that preserve specific classes of properties of the concrete state spaces of the nets. Specific properties relying on the linear structure of the concrete space state (linear time temporal logic properties (LTL)), or properties relying on its branching structure (branching time temporal logic properties (branch constraint LTL)).

State class graph: Detail information can be found in section 2.1.4.3. State class graph preserves markings, as well as traces and complete traces of the state graph, and so is suitable for reachability analysis and LTL model checking.

However, *GraphLab* is not a "model-checker" in the sense that it cannot be used to check satisfaction of a particular concrete property. *GraphLab* would be typically used as a front-end for a model-checker, providing it reduced state spaces on which the desired properties can be checked more efficiently than on the original state space (when available).

After finishing input nets in graphical format, user press button  to construct state class graphs. Outputs graphs in human readable form for available

model checkers.

4.3.2 Linear state classes, construction state class graph

Concerning Time Petri nets, state class graph construction is provided by *GraphLab*. User can set different parameters like in figure 4.9.

- *Number of States to compute:* Select show all states or number of states hope to get.
- *Visible transitions/Places:* Point out selection of show all transitions/places or number of it hope to get.
- *Properties:* LTL (Bound domains or Clock domains); Clock CTL
- *Compression:* (Relaxation, Inclusion)

Focus on *linear state classes* [BV03]. The set of linear state classes of a TPN is finite if and only if the TPN is bounded. State class graph preserving LTL formulas (bound or clock domains) and clock CTL formulas (see in Figure 4.17), since, by construction, the state class graph is deterministic.

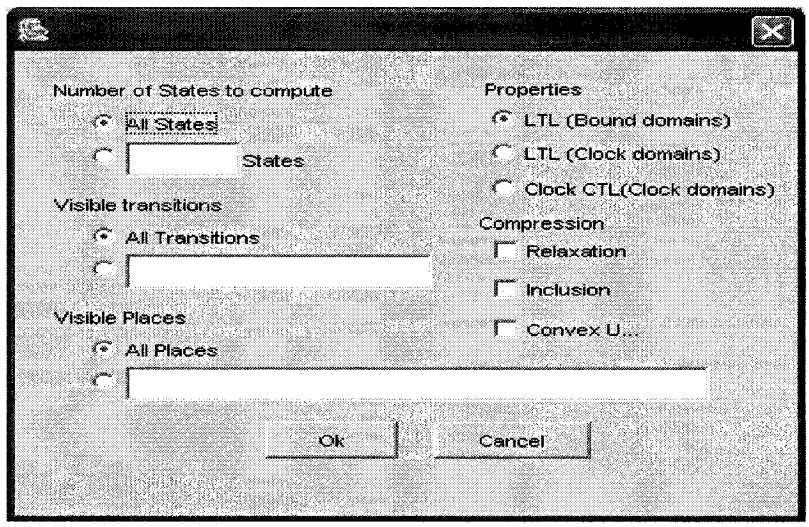


Figure 4.17 States Computation of flat level of 5 Dining Philosophers

Five dining philosophers example of figure 3.20 is a flat level. Figure of 4.9 and 4.10 are parent and child levels of hierarchy. All modification implemented in *GraphLab* must not change attributes of the time Petri nets system. Or say, the same state class graph will be got using the flat level example or hierarchical composition example. The States information is shown in Figure 4.18 and State Class Graph of 5 dining philosophers can be obtained (Figure 4.19).

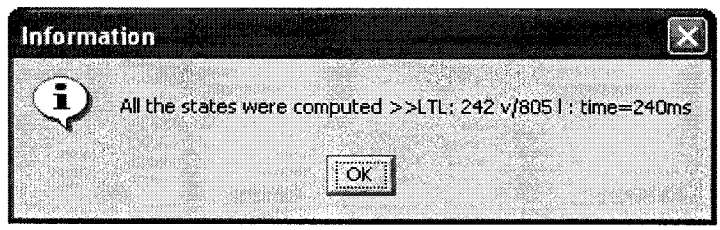


Figure 4.18 Five Dining Philosophers state class graph generation result.

(242 vertices and 805 links in state class graph)

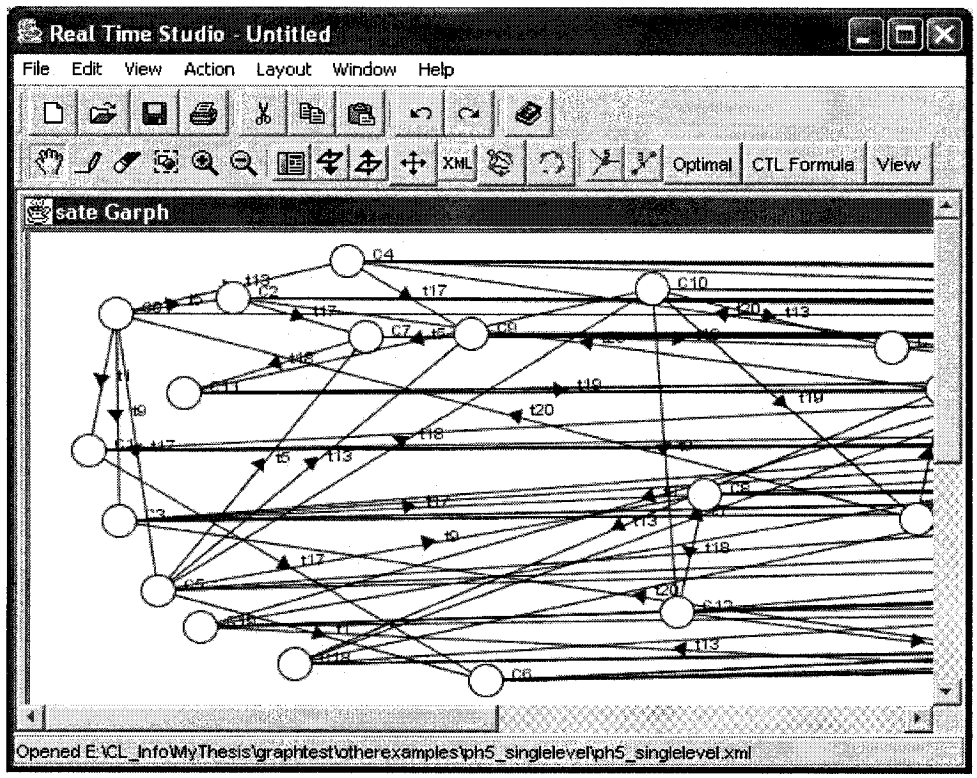


Figure 4.19 State class graph view of 5 dining philosophers

Besides bound domains, another is clock domains are select option when hope to construct state class graph. With any firing schedule, one may associate a clock function with each transition enabled at the marking reached, the clock function associates the time elapsed since that transition was last enabled. Clock domains parameter in *GraphLab* identifies an important class of time Petri nets for which clock systems exactly characterize states sets: the sets in which all transitions have bounded static intervals.

CHAPTER5

COMPLEX CASE STUDY IN MANUFACTURE SYSTEM

In this chapter a complex case study is used to investigate the capabilities of the Graphical editor for Hierarchical composition Time Petri nets using *GraphLab* by two aspects: Providing descriptions and experimental results. It show how small parts (or subnets) of a large system may be transformed by using the concept of hierarchy and the State Class Graph approach preserving linear time logic.

5.1 Introduction

In this chapter, we select a complex hierarchical composition Petri nets example in Flexible Manufacturing System (FMS). FMS is a typical real-time concurrent system composed of a number of computer-controlled machine tools, automated material handling and storage systems that operate as an integrated system under the control of host computer(s). The growing demand for higher performance and flexibility in these systems and real-time decision make to pose a significant challenge in FMS design. A formal engineering approach that helps handle the complexity and dynamics of FMS modeling, design and analysis is needed. A Hierarchical composition architectural specification model and its application in the modeling of flexible manufacturing system (FMS) are presented [ZD93], in particular hierarchy with Time Petri nets (TPN) and Real-Time Tree Logic, to form an integrated system model for architectural specification and analysis of real-time concurrent systems such as FMS [WD99].

To demonstrate the Hierarchical Compositional TPNs design methodology, this thesis adopts an example in manufacturing system with time extended from [ZD93, RA96]. The manufacturing system is show in Figure 5.1. It is composed of three workstations, W1, W2 and W3 and a robot R. A part needs to be processed by W1 first, then by W2 and finally by and finally by W3. Workstations W1 and W3 can each process only one part, while W2 can process two parts simultaneously. W1 and W3 need to use the robot during the load, processing and unloading operations, while W2 can load and unload itself. The robot is thus shared by W1 and W3. Once the robot starts loading either of the workstations, it cannot be interrupted until the job is unloaded. Fixtured parts await processing at the input storage area, INBIN. Final products will be automatically transferred to the output storage area, OUTBIN, and the fixtures will be released to the input storage area by the robot as soon the robot completes the unloading of W3. Form a Hierarchical Composition nets point of view, there are four major steps identified:

1. Operations involving input storage area: W1 begin to be scheduled after INBIN parts are available;
2. Operations on workstation W1: Robot R loads an input part from INBIN to W1→W1 processes→R unloads→W1→ moved to W2 automatically;
3. Operations on workstation W2: W2 processes the part and unloads the processed part;
4. Operations on workstation W3: Robot R loads to W3→W3 processes→R unloads W3; Output to the OUTBIN→ returns the fixture to the INBIN.

The following may be observed at the highest level of system description:

– W1, W2 and W3 operate in a sequential order on the fixtured part. If we omit the details of any specialized processing involved in each stage, then all the

workstations perform some operation on the parts, in sequence. This phenomenon can be abstracted by a single subnet place px in a hierarchical composition nets design. Workstations W1 and W3 utilize robot R while performing their operations; therefore, the acquiring of robot R has to be performed before entering px for either of these workstations.

– Robot R transfers parts from area INBIN to W1, or from the output of W2 to input of W3, or fixtures from W3 to INBIN. This process can be abstracted at a higher level as a robot "move" operation

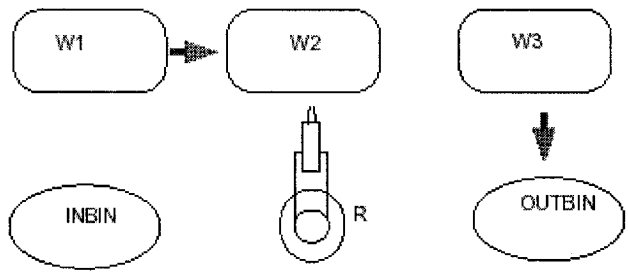


Figure 5.1 A Manufacturing Example of Hierarchical Compositional Petri Nets

The hierarchical composition (from low level to high level) structure design and the detailed example of the hierarchical model decomposition (from high level to low level) is shown in Figures 5.2, 5.3 and 5.4. The classification (Atomic and Abstract) of place and transition as detailed in Chapter 3 (Figure 3.1).

This FMS example is most abstractly constructed to a hierarchical composition time Petri nets according to their functions. When we decompose this net, a state where the grouping of places and transitions also is observed in detail. The table in Figure 5.6, 5.7 and 5.8 give the description of the places and transitions of the decompositions.

The complex system model can be decomposed into up-bottom levels. Each

level of systems description is denoted in the following: The highest level is a robot "move" operation (Figure5.2): The lower-level can be abstracted by a single subnet place "pa"(a means abstract) in a Hierarchical Compositional TPNs design. Workstations W1 and W3 utilize robot R while performing their operations; therefore, the acquiring of robot R has to be performed before entering "pa" for either of these workstations.

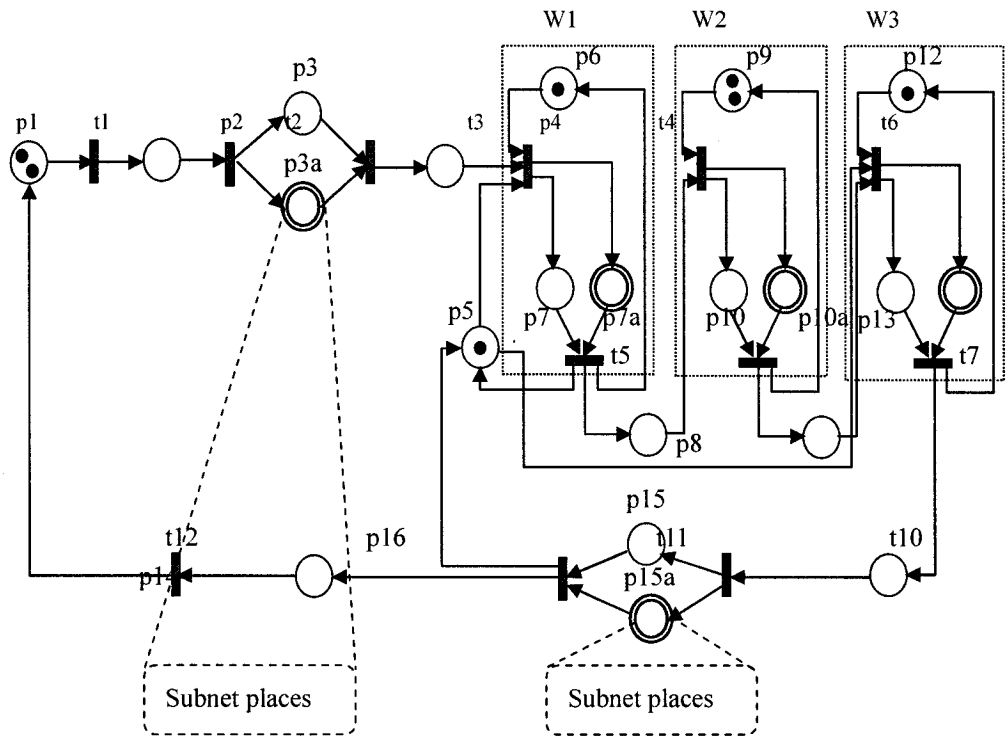


Figure 5.2 The highest level is a robot "move" operation

Abstract places p3a and p5a have the same subnet in figure 5.2.

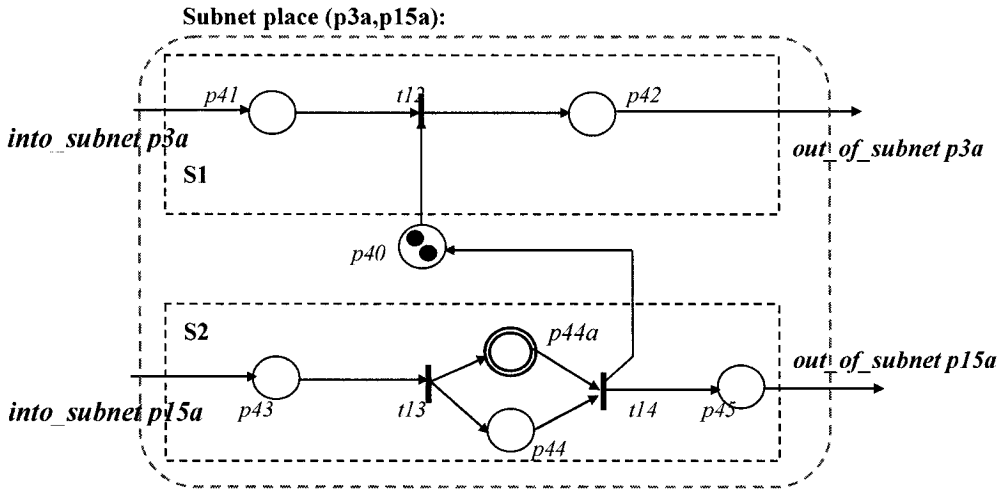


Figure 5.3 Hierarchical Time Petri Net Model Decomposition –*II.1*

These places are essentially places that abstract interacting subnets. Therefore, they have 2 points of entry and exit. A token flowing out of subnet p3a enters subnet p15a after some time. Thus, by looping p42 with p43 after inserting an appropriate delay, this interacting subnet can be verified for properties. Abstract places p7a, p10a, p13a have another same subnet in figure 5.4. Like p3a and p5a (figure 5.3), they construct the second lower-level of system.

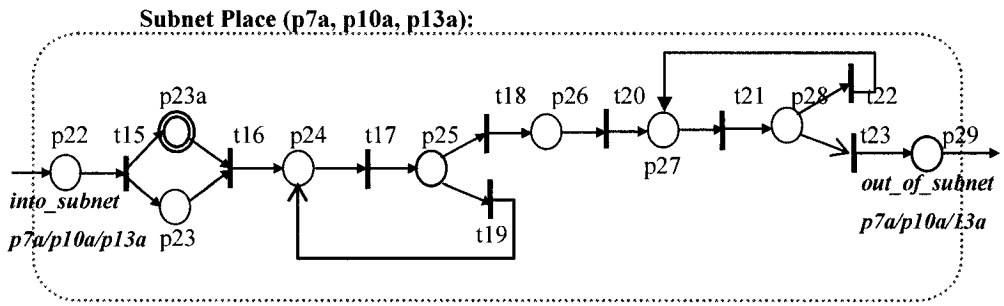


Figure 5.4 The Hierarchical Time Petri Net Model Decomposition –*II.2*

Abstract place p23a is the subnet of p7a, p10a, p13a; Abstract place p44a is the subnet of p3a and p5a. p23a and p44a have the same structure (figure 5.5), they

are the lowest-level in this manufacturing system.

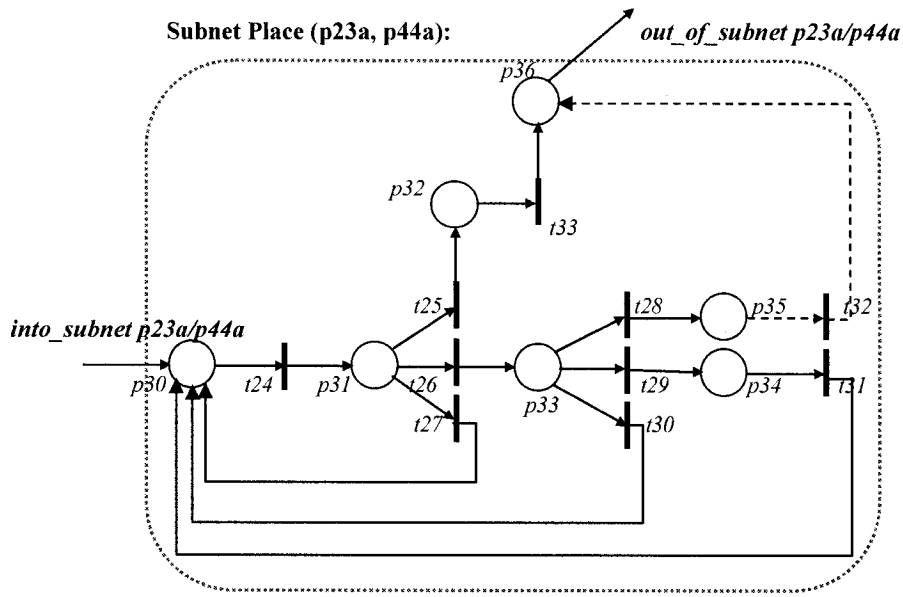


Figure 5.5 Hierarchical Time Petri Net Model Decomposition–*III*

Places descriptions are in the figure 5.6. Description of abstract elements and subnets describe are in figure 5.8. Transitions with Time Interval values of transition are gave in the figure 5.7.

PLACE	DESCRIPTION
P1	Input *ss* place
P2	Input part available
P3	AbstractPlace for number of fixtures
P4	Fixtured part available at INBIN
P5	Robot free
P6	W1 (Workstation 1) free
P7	AbstractPlace for operations at W1
P8	Part unloaded from W1
P9	W2 (Workstation 2) free
P10	AbstractPlace for operations at W2
P11	Part unloaded from W2
P12	W3 (Workstation 3) free
P13	AbstractPlace for operations at W3
P14	Part unloaded from W3
P15	AbstractPlace to return fixture to INBIN
P16	Processed part at OUTBIN
P40	Number of fixtures available at INBIN
P41	Into AbstractPlace p3
P42	Out of AbstractPlace p3
P43	Into AbstractPlace p15
P44	AbstractPlace to move fixture to INBIN
P45	Out of AbstractPlace p15
P22	Begin operations at W1, W2 or W3
P23	Move part to W1, W2 or W3
P24	Load part at W1, W2 or W3
P25	Load operation successful?
P26	Load successful, Process part
P27	Unload part from W1, W2 or W3
P28	Unload operation successful?
P29	Part unloaded, out of AbstractPlace
P30	Into move AbstractPlace, move a small distance
P31	Payload variations / Destination reached?
P32	Initialize to put down part
P33	Part dropped
P34	Initialize and pickup part
P35	Part out of reach, hard failure
P36	Out of move AbstractPlace

Figure 5.6 Descriptions of Places

RANSITION	DESCRIPTION	FIRING INTERVAL
T1	EDT 1	[0,inf]
T2	Into AbstractPlace p3	[0,inf]
T3	Out of AbstractPlace p3	[0,inf]
T4	Begin W1 operations	[0,inf]
T5	End W1 operations	[0,inf]
T6	Begin W2 operations	[0,inf]
T7	End W2 operations	[0,inf]
T8	Begin W3 operations	[0,inf]
T9	End W3 operations	[0,inf]
T10	Into AbstractPlace p15	[0,inf]
T11	Out of AbstractPlace p15	[0,inf]
T12	Fixure available for input	[0,inf]
T13	Into move AbstractPlace	[0,inf]
T14	Out of move AbstractPlace	[0,inf]
T15	W1, W2 or W3 operations intiated	[0,inf]
T16	Part moved to workstation	[0,inf]
T17	Loading completed	[0,inf]
T18	Loading successful	[0,inf]
T19	Loading unsuccessful	[0,inf]
T20	Part processed	[0,inf]
T21	Unloading completed	[0,inf]
T22	Unloading unsuccessful	[0,inf]
T23	Unloading successful	[0,inf]
T24	Part moved a small distance	[0,inf]
T25	Destination reached	[0,inf]
T26	Payload variation	[0,inf]
T27	Destination not reached	[0,inf]
T28	Part dropped out of reach	[0,inf]
T29	Part within reach	[0,inf]
T30	Part not dropped	[0,inf]
T31	Part picked up	[0,inf]
T32	Part can be rejected	[0,inf]
T33	Part placed at destination	[0,inf]
Ti	loop back	[0,inf]

Figure 5.7 Description of Transitions and Time Interval

PLACE	DESCRIPTION
P3a	Abstract Places in the highest level which represent a same subnet that include an abstract place p44a.
P15a	
P7a	Abstract Places in the highest level which represent a same subnet that include an abstract place p23a.
P10a	
P13a	
P23a	Abstract Places in the lowest level which represent a same subnet that include no any other abstract elements.
P44a	

Figure 5.8 Descriptions of Abstract Places and Subnets

5.2 Hierarchical Structure Model Development Design and Specification

In this section we implement construction for this FMS example by hierarchical TPNs, shown in Figure 5.9.

The objective of this effort has been two fold:

- To demonstrate hierarchy is one of useful classifications in the theory that help to built compositional structures.
- To apply the hierarchical composition and decomposition mechanism and their ability can cope with large time Petri nets in time extended systems.

Connection points and transfer tokens between high and low level are key sides when design a complex systems. Figure 5.10 and Figure 5.11 are connection information in *Attributes viewer* of *GraphLab*. Figure 5.10 shows arc connection information between the 2^{th} and the 1^{st} level; Figure 5.11 shows arc connection information between the 3^{th} and the 2^{th} level.

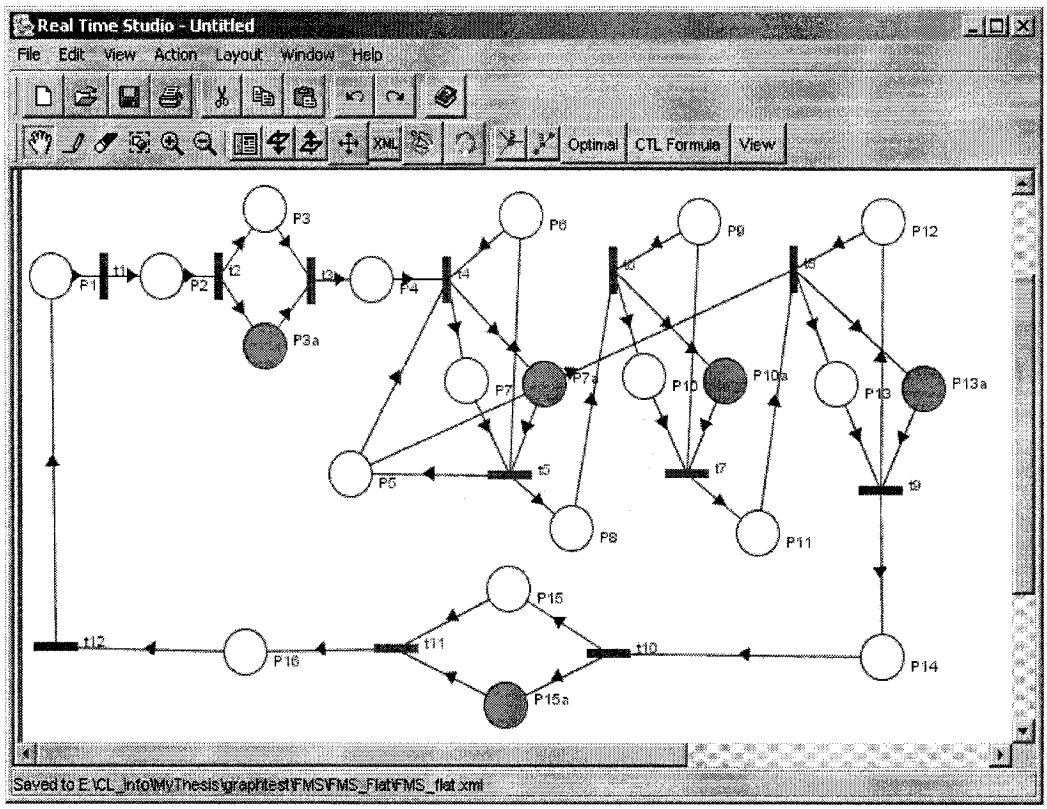


Figure 5.9 Hierarchical design of example in *GraphLab* -1st level

ID	supernetID	Name	Value
p41	p3a	connectFrom	t2
p42	p3a	connectTo	t3
p43	p5a	connectFrom	t10
p45	p5a	connectTo	t11
p22	p7a/p10a/p13a	connectFrom	t4/t6/t8
p29	p7a/p10a/p13a	connectTo	t t5/t7/t9

Figure 5.10: Arcs connection information between 2rd and the 1st level

ID	supernetID	Name	Value
p30	p23a/p44a	connectFrom	t15/t13
p36	p23a/p44a	connectTo	t16/t14

Figure 5.11: Arcs connection information between 3th and the 2rd level

CHAPTER 6

CONCLUSION

This project presents a research of current existing compositional techniques for large models. And find an approach provides the necessary concepts of hierarchy and abstraction which are effective for analysis and are obvious requirements for the construction of the time extended model – time Petri nets. In the end, a software tool is developed or improved to realize this kind of compositional technique.

In this chapter, we first recall our objectives and then review the syntheses of research and contributions, limitations of current status, and future works.

6.1 Research Objectives

There are three primary objectives in this thesis:

Review existing approaches in compositionality area of Petri nets and offer an up-to-date detailed survey of these approaches, classify explicitly composition mechanisms.

Present an effective way to construct time Petri nets after taking advantages of recent development approaches among these various composition techniques for Petri nets with time extended systems. This approach that is selected or developed in this thesis is not only necessary for analysis but is obvious requirements for the construction of many sizeable models.

Implement this compositional structure construction by textual or graphical editor of a software tool that has been improved and programmed in this project.

6.2 Syntheses of Research and Contributions

This research has led to the following contributions:

Presents a precise notion of composition and investigate, review existing approaches in compositionality area of for large Petri nets models, in particular, in the area of Petri nets with time extended system.

Comprehensively describes import concepts about composition from elementary to those advanced: *Connector, Component (Subnet, Supernet), Fusion, Folding, Abstract / Atomic, Hierarchy / Refinement, Algebra, Object-oriented.*

Besides this project offers an up-to-date detailed survey for current existing compositional approaches, to more advanced points according to literatures review, this project classifies compositional methods: Petri nets composition structuring can be classified into 3 primary types:

- ***Modularization Composition*** with *object orientation* technique;
- ***Horizontal Composition*** with *place/transition sharing, folding* and *reduction techniques (place/transition fusion, component-level reduction)*;
- ***Hierarchical Composition*** with *abstraction* or *refinement* technique;

Besides above three primary structuring methods types, there are still some composition methods like *algebra* (pBC, M-nets, Well-formed net), *work-flow, trace theory*, they are also introduced summarily and compactly in this project.

Put forward satisfactions requirements after compositionality structuring, the equivalent compositional nets will be got without affecting any significant properties of the whole model (*behavior and properties equivalences*). No matter horizontal, hierarchical or modularization technique using for compositionality, the composed system specification may look like the whole net (*whole net likeness*).

As a result, an effective compositional approach is presented to construct time Petri nets – *hierarchical compositional technique*. A top-down hierarchical design procedure for obtaining a compositional model for large systems has different levels topological structure. They have been proposed and explained for describing the functionality of the coordination level of a hierarchical system. This kind of compositionality can re-use the information about the sub-components of a component to obtain information about the component's behavior in models. However, current approaches, though dealing with the concept of hierarchy through sound formalisms, are not completely appropriate for time extended systems since the classical PNs model lacks essential notions like timing. An important contribution of this thesis is the definition of hierarchy for a modeling formalism suitable for the design and verification of Time Petri Nets.

Significant contribution in this project is development of the convenient graph editor approach with graphical user interface which is programmed in this thesis based on *GraphLab*. The *GraphLab* Petri net editor has provided improved support for hierarchical Petri nets; connection interaction points can easily be selected / deselected. The dining philosophers and FMS example (chapter 3, 4, 5) demonstrate the advantage of designing complex systems using a compositional approach, especially if the system contains repetitive units, like the philosophers in this example.

Improved parts in *GraphLab* editor allows subnets to be added to any other Petri Nets, allowing multiple level Petri nets to be designed, it help designers construct TPNs by a composing a number of fully understandable entities instead of flat representation, components and functions added in *GraphLab* can transfer tokens between different levels in hierarchy, can define abstract places or transitions. To move down a level, select the *expand* icon, subnets can then be viewed by clicking on it. To move up a level a single click on the back level icon (*collapse* icon) is required. Connection points with subnets or supernets can be visibly seen on the attribute viewer list. This feature enables arcs between levels to easily be specified and observed.

Last, the usual way to add hierarchical composition to Petri nets is the refinement of either or both of its standard atomic building blocks (places and transitions). A common problem is that refining places or transitions destroys some of their key properties. In this project we will therefore use an approach based on a structuring concept of high-level Petri nets that was developed. Instead of trying to directly refine either places or transitions, we have considered components to be subnets with inputs and outputs, which can be embedded into other components. That is the reason why we can generate the same state class graph view in *GraphLab* preserving LTL properties for both flat level and hierarchical composition structure models.

6.3 Limitations of Current Work

A crucial characteristic of hierarchical conceptual drawings is that they have function of re-use and abstraction. For example, 5 dining philosophers and FMS cases used in this project both have this re-useable characteristic. Thinking, picking chopsticks, eating, and releasing chopsticks are four states for every philosopher. So,

GraphLab editor can only draw one subnet file and copy them to other four directories, and modify their connection parameters (connectTo, connectFrom in *attributes viewer list*). The complex case of FMS in chapter 5 can take advantage of abstraction concept of hierarchy enough is the reason of the re-useable ability. (i.e., p3a and p5a (Figure 5.3) have same firing transitions, places and tokens transformation between places. When after designing p3a's subnet, it can be copied to p15a's directory. Same thing happens on p7a, p10a, p13a group (Figure 5.4); p23a and p44a group (Figure 5.5)). Thereupon, the most limitation of the hierarchical composition TPNs is it can not be available for all large Petri nets models. Those models which is lack of common characters or same firing information for some parts of places/transitions, using abstraction and refinement between levels is not possible. In other words, if those Petri nets can not be grouped into different parts according to their firing states, and if theses parts (or subnets) can not be decomposed from top to bottom, this large system have low efficiency while using hierarchical composition from bottom to top.

Generally speaking, re-useable or common firing information (places / transitions / token transformation) from run function is the precondition of good using re-useable ability in this hierarchy approach and structure edit in *GraphLab* software tool.

State Class Graph is an enumerative technique for TPNs used in *GraphLab*, it preserves markings, as well as traces and complete traces of the state graph, and so is suitable for reachability analysis and LTL model checking.has to be built and the observer can dramatically increase the size of the state space. However, for Petri nets with size a little large, we will generate amount of vertices and links when drawing state class graph. In the case of dining philosophers, if setting 3 philosophers, all the states were computed are 27 vertices, 54 links. If setting 5 philosophers, all the states

were computed are 242 vertices, 805 links. At this rate, state class graph is illegible when we view it in *GraphLab*.

6.4 Future works

Some software improvements and Modular analysis and incremental analysis are future works that will be illuminated in this section.

Software *GraphLab* improvements are necessary for *GraphLab* in order to construct hierarchical structure smoothly:

- The implementation enables single mouse clicks to move between different levels of a hierarchical Petri net. Navigation browser by mouse clicking could be added in the future. This function will let *GraphLab* for hierarchical manipulation more convenient.

- Connection points parameters are key characteristics in *GraphLab* Petri net editor has provided improved support for hierarchical Petri nets; connection description for input or output between parent and child level can easily be selected/deselected by string format (i.e., p1, p2, ...). But, any error input has great influence on the outcome of the design. Only accurate sting format in this filed can get right results. Programming modification selection input or output elements from background by COMBOX will eliminate this error hand-made. This modification need to create a file to record subnetID and supernetID information dynamically by reading XML file (data for new added characteristics about hierarchy in *attributes viewer* get from XML file). This work will be done in the future.

Modular analysis and incremental analysis are two important future works.

Modular analysis will reduce the size of the state space and be particularly suited to systems with strong cohesion for subnets and supernets or mutual-subnets. Modular analysis is one of our future works that because complex Time Petri Nets consists of a number of semiautonomous subnets. Subnets always have significant local behaviors, these behaviors properties occasionally interrupted by other subnets. Future Modular analysis approach will explore the local behavior without considering all the possible, at the same time, Modular analysis will explore the local behavior of other subnets. Our future modular analysis of time Petri Nets will be further facilitated by the observation that subnets (or subsystems) and determine exactly subnets and supernets connection relationship.

Another future work of modular is the desirable modular verification will would be used to check the properties of the modules separately and infer those of the whole system. Verification based on a modular approach is not straight forward and led to several research trends in the future. Several techniques have been proposed to push further the use of modularity for verification purposes.

Because the design of compositional time Petri nets in this project is incremental: they are done at an abstract level firstly, and then parts of it are refined. So, incremental analysis is also the key in future verification study of large and complex systems. Subnets are not completely considered in isolation: the other ones are abstracted away by abstraction places. Thus, when analyzing a module, information about its environment behavior is present as well. The abstraction places/transitions are determined through invariants computation for the complete system. Our compositional TPNs methodology is based on the theory of abstract

interpretation. Our future incremental methodology will be proposed for the analysis and verification of Compositional Time Petri nets (TPNs). An appropriate incremental analysis way will generate separately for each subnets (or subsystems). In the future, incremental analysis will be asked to propose sufficient conditions for deducing properties of the whole Petri net from those of its subnets (or subsystems).

BIBLIOGRAPHY

[AC93] R. Alur, C. Courcoubetis, and D. L. Dill, *Model checking in dense real-time*, Information and Computation, 104(1):2-34, 1993.

[AN93] Valmari Antti, *Compositional state space generation.* Advances in Petri nets.1993.

[AC01] G.A.Agha, F.De Cindio, G.Rozenberg, *Concurrent Object Oriented Programming and Petri Nets,* Advances in Petri Nets, Lecture Notes in Computer Science, 2001

[BB02] H.Boucheneb, G.Berthelot, *Contraction of the ITCPN state space,* Electronic Notes in Theoretical Computer Science, April 2002.

[BE83] G. Berthelot. *Transformations et analyse de r'eseaux de Petri: Application aux protocoles*. PhD thesis, Universit'e de Paris 6, 1983.

[BF95] E.Best, H. Fleischhack, *A Class of Composable High Level Petri Nets,* Proceeding of the 16th International Conference on Application and Theory of Petri Nets, Turin, June 1995

[BH04] H.Boucheneb, R.Hadjidj, *Using inclusion abstraction to construct atomic state class graphs for Time Petri Nets*, International Journal of Embedded Systems, accepted in 2004.

[BH05] H.Boucheneb, R.Hadjidj. *Much Compact Time Petri Net State Class Spaces*

Useful to Restore CTL Properties,* Fifth ACSD's conference, pp. 224-233, 2005.

[BB98] H.Boucheneb, G..Berthelot, *Composition de réseaux de Petri temporels,* Techniques et Sciences Informatiques, Volume 17, n°6, pp 671-698, 1998.

[BD91] B. Berthomieu, M.Diaz, *Modeling and verification of time dependent systems using time Petri nets*, IEEE Transactions on Software Engineering, 17(3), 1991.

[BE01] B. Berthomieu, *La méthode des Classes d'états pour l'Analyse des Réseaux Temporels - Mise en Oeuvre, Extension à la multi-sensibilisation, Modélisation des Systèmes Réactifs*, MSR'2001, Hermes, 2001.

[BF95] Eike Best, Hans Fleischhack, *A Class of Composable High Level Petri Nets.* Application and Theory of Petri Nets. 1995.

[BM83] B.Berthomieu and M. Menasche, *An enumerative approach for analyzing time Petri nets*, in *Proc. IFIP Congr.*, Paris, France, Sept. 1983.

[BA02] P. Bonhomme, P. Aygalinc, G. Berthelot, S. Calvez, *Hierarchical control of time Petri nets by means of transformations.* Hammamet, Tunisia, Vol4 IEEE Computer Society Press, October 2002.

[BO01] K.Bohuslav, *Analysis of Object-Oriented Petri Nets,* 4th International Conference on Information System Modelling - ISM'01, H Republic, CZ, MARQ, 2001

[BV95] G.Bucci, E.Vivario, *Compositional validation of time-critical systems using communicating time Petri nets*, IEEE Trans. Softw. Eng.,V21, no. 12, 1995.

[BV03] B.Berthomieu, F.Vernadat *State Class Constructions for Branching Analysis of Time Petri Nets*, Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003), pages 442-457. Springer Verlag, February 2003.

[CE01] Luis Alejandro Cortes, Petru Eles, Zebo Peng, *Hierarchical Modeling and Verification of Embedded Systems,* Euromicro Symposium on Digital Systems Design, Warsaw, Poland, Sept. 4-6, 2001.

[CO95] C.J. Coomber, *Petri Net Composition with Trace Theory*, Report of School of Computing and Mathematics, Deakin University. 1995

[CP00] S.Christensen, L.Petrucci, *Modular analysis of Petri nets*, Computer Journal, Vol. 43, No. 3, pages 224-242. 2000.

[ES91] J.Esparza, M.Silva, *Top-down synthesis of live and bounded free choice nets*, Advances in Petri nets,1991.

[ES96] R.Esser. *An Object Oriented Petri Net Approach to Embedded System Design.* PhD thesis, ETH Zurich, 1996.

[FC01] P.Franck, S.Christian, *FIFO buffers in hot tie sauce,* LACL 04-2001.

[FF91] G. Florin, C. Fraize, and S. Natkin, *Stochastic Petri nets: Properties, applications and tools*, Microelectron. Reliab. V31, no. 4, pp. 669–697, 1991.

[GL81] H. J. Genrich, K. Lautenbach. *System Modelling with High-Level Petri Nets*, Theoretical Computer Science, V13. North-Holland, 1981.

[HA72] M. Hack. *Analysis of production schemata by Petri nets.*Technical report. MIT. Boston 1972

[HB98] J.Hong, D.Bae, *HOONets: Hierarchical Object-Oriented Petri nets for System Modeling and Analysis,* Technical report, Korea Advanced Institute of Science and Technology, November 1998.

[JA98] W. Janneck, *Compositional Petri net structures, Introduction and formal definition,* TIK Report 60, November 1998

[JE92] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, Basic Concepts of EATCS Monographs in Computer Science.V1, Springer-Verlag, 1992.

[JL81] James Lyle Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice Hall PTR. 1981

[KA86] A.Kaldewaij, *Formalism for Concurrent Processes*, Ph.D. Thesis, University of Technology Eindhoven, 1986.

[KM03] M.Koutny, *A Compositional Model of Time Petri Nets*, Lecture Notes in Computer Science, Volume 1825 / 2000 June 2003

[KP00] H.Klaudel, F.Pommereau, *A Concurrent and compositional Petri net*

semantics of preemption, Proc.of IFM'00,LNCS 1945 P 318-337, Springer,2000.

[LA95] Charles A. Lakos. *From coloured Petri nets to object Petri nets.* In Proceedings of the 15th International Conference on Applications and Theory of Petri Nets Lecture, Notes of Computer Science, 1995.

[LK94] C.A.Lakos, C.Keen, C. *LOOPN++: A New Language for Object-Oriented Petri Nets,* Computer Science Department, University of Tasmania, 1994

[ME74] P. Merlin. *A Study of the Recoverability of Computer Systems.* Thesis in Computer Science Dept, University of California, 1974.

[MU89] T. Murata, *Petri Nets: Analysis and Applications*, in *Proc. IEEE*, vol. 77, pp. 541-580, April 1989.

[PF03] E. Pelz,H. Fleischhack. *Compositional high level petri nets with timing constraints - a comparison.* In *Proceedings ASCD*, pages 132-141. IEEE Press, 2003.

[RA74] C. Ramchandani, *Analysis of asynchronous concurrent systems by Petri Nets*, Cambridge, MA: MIT, Project MAC, TR-120, Feb. 1974.

[RA96] S. Ramaswamy, K. P.Alavanis, *Hierarchical Time-Extended Petri nets (H-EPNs) based error identification and recovery for multilevel systems*, IEEE Transactions on Systems, Man and Cybernetics, pages 164-175. 1996.

[RP02] J.Richling, L.Popova-Zeugmann, M.Werner, *Verification of non-functional*

properties of a composable architecture with Petri nets, Fundamenta Informaticae, 51, 2002.

[RW02] J.Richling, M.Werner, L.Popova-Zeugmann, *Automatic Composition of Timed Petrinet Specifications for a Real-Time Architecture*, Proceedings of IEEE International Conference on Robotics and Automation, Washington DC, 2002.

[RS83] M. Rem, Van De Snepscheut, L. A., and Udding, J. T, *Trace Theory and the Definition of Hierarchical Components*, in Third Caltech Conference on VLSI, pp. 225-239, Computer Science Press, 1983.

[SA97] Anil Sawhney, *Petri Net Based Simulation of Construction Schedules.* Simulation Conference, 1111-1118, 1997.

[SB93] C.Sibertin-Blanc, *A client-server protocol for the composition of Petri nets*, In Proceedings of the14th International Conference on Applications and Theory of Petri nets. June 1993.

[SB96] R. Sloan , U. Buy, *Reduction rules for time Petri nets*, Acta Inform., vol. 33, pp. 687–706, 1996.

[SM83] I. Suzuki, T. Murata, *A Method for Stepwise Refinement and Abstraction of Petri Nets*, in Journal of Computer and System Sciences, vol. 27, pp. 51-76, Aug. 1983.

[SO91] Y.Souissi, *A modular approach for the validation of communication protocol using FIFO nets.* 1991

[TM04] L.Timo M.Mako, *LTL Model Checking for Modular Petri Nets.* Proceedings of Applications and Theory of Petri Nets 2004

[TY95] J. Tsai, S.Yang, and Y. Chang, *Timing constraint Petri nets and their application to schedulability analysis of real-time system specifications, IEEE Trans. Softw. Eng.*, vol. 21, no. 1, pp. 32–49, 1995.

[VA79] R.Valette, *Analysis of Petri Nets by Stepwise Refinement,* in Journal of Computer and System Sciences, vol. 18, pp. 35-46, Feb. 1979.

[VA94] R.Valette, R, *Hierarchical High Level Petri Nets for Complex System Analysis*. Computer Science, Vol. 815, Application and Theory of Petri Nets 1994

[WR90] W. Brauer, R. Gold, and W. Vogler. *A survey of behaviour and equivalence preserving refinements of Petri nets.* In Advances in Petri Nets.1990

[WD99] J.Wang, Y.Deng, *Incremental modeling and verification of flexible manufacturing systems*, J. Intell. Manuf., vol. 10, no. 6, pp. 845–502, 1999.

[WD00] J.Wang, Y.Deng, *Compositional Time Petri Nets and Reduction Rules.* IEEE transactions on systems, man and cybernetics-part B: cybernetics, V30, No.4. August 2000.

[ZD93] M.C.Zhou, F.Diceasare, *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*, Kluwer Academic Publishers, Boston, 1993.

[ZH95] M.C.Zhou, *Petri Nets in Flexible and Agile Automation*. Norwell, MA: Kluwer, 1995.

[LINK] http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html